Coventry
University

Coventry University Repository for the Virtual Environment (CURVE)

Tessa Daniel

Investigating into the use of advance sensing technologies for protection suits.
Published Report (PDF)
(Refereed)

Daniel, T. (2008) *Information extracting from large-scale WSNs – A complete querying perspective*. Report no. COGENT.002 Coventry: Coventry University

http://curve.coventry.ac.uk

COGENT
computing

Information Extraction from Large-scale WSNs - A Complex Querying Perspective

Tessa Daniel
t.daniel@coventry.ac.uk

Regardless of the application domain and deployment scope, the ability to retrieve information is critical to the successful functioning of any wireless sensor network (WSN) system. In general, information extraction procedures can be categorized into three main approaches: agent-based, query-based and macroprogramming.Of the three, query-based systems are the most popular mainly because they provide a usable, high level interface to the sensor network while abstracting away some of the low level details like the network topology and radio communication. In contrast, macroprogramming provides a more general-purpose approach to distributed computation compared to traditional query-based approaches and focus on programming the network as a whole rather than programming the individual devices that form the network. The agent-based approach tailors the information extraction mechanism to the type of information needed and the configuration of the network it needs to be extracted from. This report surveys extensively the literature in the area of WSN information extraction, highlights the benefits of in-network processing and proposes a novel hybrid approach that incorporates query and macroprogramming techniques for information extraction in large-scale, informational systems. The feasibility of the approach is demonstrated through simulation.

This is an interim report of work covered in the first year of the PhD program entitled as above.

Coventry University

Contents

List of Figures

Chapter 1

Introduction

The increase in availability and affordability of wireless technology has led to a proliferation of large wireless sensor networks (WSNs) with increasing numbers of nodes deployed to resolve complex "informational" problems. Typically, however, nodes in these networks have limited resources including energy (given the limited battery power), memory, computing power and communication bandwidth. The coupling of high complexity global tasks and the reduced resource nodes make WSNs a rich research domain. The use of WSNs has been explored in a number of application domains with deployments ranging from scientific research to battlefield surveillance for the military. Some examples of deployments include habitat monitoring applications [73, 44]; agricultural monitoring [64]; healthcare [35]; disaster management and detection [19] and military applications [92]. Moreover, some newer applications involve the integration of multiple WSN systems spanning a variety of disciplines. The High Performance Wireless Research and Education Network [52] and ROADNet [104]projects, both in California, are two examples of such multidisciplinary applications currently in operation.

Although specific application requirements may differ from one WSN system to another, essential to all WSN systems is the ability to acquire data and generate information in order to fulfil the application needs. Many of the WSN information extraction approaches currently in operation are based on applicative query mechanisms where requests are initiated via queries written in an appropriate declarative language, posed to the network, and data or information generated as a result. Data processing in such applications usually follows one of two approaches: centralised or distributed. For some systems, resources are not as severely constrained so minimising energy usage is not a major concern. In such unconstrained systems, a centralised approach to processing is often used where sensed values are pushed to a power-rich location for processing, which may involve cleaning and querying the data as part of more in-depth analyses.

In a large number of systems, however, constraints like computing and battery power dictate that applications be developed with energy-efficiency in mind. For many of these applications human intervention can be both time consuming and expensive, for example, in terms of changing batteries or adding new nodes. Therefore, a key design objective is to extend the lifetime of the network as much as possible. It is well-understood that power usage costs in WSNs are dominated by communication as opposed to computation [48, 70, 94]. Therefore, techniques which promote a decrease in communications while using

in-network data reduction have been identified as key to creating more energy-efficient WSN applications. Distributed processing is proposed as a method that promotes processing of data on nodes in the network [70, 11]. This in-network processing takes advantage of nodes' processing power and may include techniques like data aggregation, fusion or elimination of redundant values through filtering [132, 47]. The net result is more energy-efficient applications since data transmission is reduced in exchange for in-network computation.

Given the apparent advantage (in terms of energy usage) of utilizing in-network processing, this PhD program is concerned with the development of novel, deployable, information extraction mechanisms that incorporate some form of in-network processing.

1.1   Information Extraction

Regardless of the application domain and deployment scope, the ability to retrieve information is critical to the successful functioning of a WSN system. In general, information extraction procedures can be categorized into three main approaches: agent-based, query-based and macroprogramming. Of the three, *query-based* systems are the most popular mainly because they provide a usable, high level interface to the sensor network while abstracting away some of the low level details like the network topology and radio communication. They are very useful and provide a solution in cases where data needs to be retrieved from the entire network. With this ease of use, however, come a number of limitations.

The first limitation is in terms of what queries can be posed to the network. In general, the query-based applicative systems in use allow the issuing of restricted queries, ranging from those targeting raw sensor readings on nodes in the network to those requiring the computation of simple aggregates like average, maximum, and minimum over some attribute of interest, for the entire network. Second, the query languages used cannot easily express spatio-temporal characteristics which are an important aspect of the data generated in WSNs. Third, it is quite difficult if not impossible to construct information requests that represent higher level behaviour, or involve just a 'subset' of the network (whether physical or logical) or require more complex in-network interactions between 'subsets' in order to generate information. Furthermore, as distributed computation is not the main focus of SQL-based query languages (which support most WSN query-based approaches) implementing arbitrary aggregation, for example, is quite difficult [87].

In contrast, *macroprogramming* has been proposed as an approach to information extraction that provides a more general-purpose approach to distributed computation compared to traditional query-based approaches. As applied to WSNs, macroprogramming approaches focus on programming the network as a whole rather than programming the individual devices that form the network. Many macroprogramming systems provide the ability to create programs that represent higher level behaviour, a level of abstraction beyond that of the more popular query-based approaches. Global behaviour can be specified, programmed and then translated to node level code. Ideally, the programmer is not concerned with low level details like network topology, radio communication or energy capacity.

Of interest with some macroprogramming systems are the application-defined, in-network abstractions (some based on local node interactions) that are used in data processing. One example is the Regiment

system [89]in which a programming abstraction called a region is used. A region is described as a collection of spatially distributed signals with an example being the set of sensor readings from nodes in a geographic area. Regions as opposed to individual nodes are programmed (for example, an rfold operator is used to aggregate the values in a region into a single signal which can then be communicated to the user or used in further computation).

Macroprogramming, however, still presents a number of challenges. First, creating a powerful macroprogram requires a learning curve for the programmer. Expressing high-level requirements are not necessarily as intuitive to the user as perhaps SQL-based approaches are. Second, although proposed as an approach that eliminates the need for node-level programming, many of the current macroprogramming systems provide node-specific abstractions, undermining the rapid development and productivity advantages macroprogramming is meant to provide. Finally, because of the wide semantic gap that exists between the high level program and the node level code, compiler construction is quite challenging. The code generated as a result of compilation has to cater for not just computation but node-level communication as well.

A third approach to information extraction looks at the extracting of information in a network-aware manner and tailors the mechanism to the type of information needed and the configuration of the network it needs to be extracted from. These models use *agents* to perform tasks, make decisions and collaborate to achieve more complicated tasks. An agent is simply a piece of software that performs a task without the need of user invocation to function. Many agent-based models are multi-agent systems in which multiple agents sometimes coded to function in different ways, collaborate, coordinate and organize to perform complex tasks and generate information. These models therefore introduce expressiveness and flexibility in terms of what functions they can facilitate in the network as well as in facilitating the ability for agents to make decisions while in the network. Agents can act autonomously, multiple agents can run on a node at the same time, and multiple applications can co-exist in the network. Mobile agents can move, clone themselves and act to deal with unexpected changes in the environment [98, 33].

Although attractive, in theory, the agent-based approach presents a number of challenges particularly in terms of the difficulty they present for non-expert users to design and program. First, given the complexity of distributed systems, it is a challenge to model accurately what interactions will be taking place between components in the network when designing agents. Second, when implementing the agents, particularly mobile agents, the code must be able to run in unpredictable, resource-constrained environments. Forecasting accurately exactly what conditions will exist in the network is difficult and therefore difficult to program for. Third, there is difficulty in testing and debugging, given the distributed nature of the system. In mobile agent systems, tracking and understanding execution in the face of agents moving from node to node makes debugging and testing even more of a challenge. Given these difficulties, practical implementations of agent-based systems, particularly multi-agent and mobile agent systems, are not as widespread as query-based systems are. Each of the three approaches above is suitable for a number of applications. They each have strengths but pose challenges as well. The agent-based approach provides a high degree of expressiveness and flexibility as do the macroprogramming approaches, but they both are more difficult to implement into deployable WSN systems. The query-based systems have, as their key feature, ease of use but are limited in the types of queries that can be posed to the network. Macroprogramming tries to address this by providing powerful constructs for capturing higher level behaviour through global programs but introduce a steep learning curve to the user.

The author argues that a hybrid approach that retains the simplicity and ease of use of the more traditional query-based approaches while allowing the inclusion of useful logical abstractions provided by macroprogramming approaches to facilitate construction and resolution of more powerful queries is possible and indeed desirable. This approach can also incorporate some of the principles of agent-based systems, such as collaboration and decision making in the network, in assisting in query resolution.

In some applications, it is necessary to not only have information generated from within the network and communicated to a requesting node but to also be able to pose questions that represent higher level requests. This class of applications provides the inspiration for this research work and is discussed in greater detail in Chapter 3. To summarize, given a motivating scenario, two issues are addressed in the proposed research programme.

First, is the desire to provide users of WSN applications with the ability to construct and pose higher level information requests instead of simple queries requiring the collection of raw sensed values or the calculation of simple aggregates over the entire network. Complex queries are proposed as suitable constructs that can allow the expression of spatio-temporal characteristics and requests requiring more involved in-network interactions to generate information.

Second, is the desire to provide a system that produces responses to these higher level requests for information within the network instead of as a result of post-collection analysis of all data.

A series of research questions have been formulated in response to these two topics. These questions, together with the proposed research methodology are presented in the following sections.

1.2   Research Questions

This PhD program proposes to investigate the methods for constructing higher-level information requests as well as techniques for processing these requests within the network. There are three main research questions:

1. Is it possible to construct and disseminate complex queries that allow a user to program the sensor network in a similar way to current applicative approaches?

2. Is it possible to create an in-network distributed query processing system that allows information to be generated within the network and make that information available to the user?

3. Does such an in-network distributed query processing system offer solutions where current centralized approaches fail?

The research questions are detailed below.

1.2.1 Complex Queries for Representation of Higher Level Information Requests

Some research has addressed the need for complex queries and for the ability to process these queries within the network. [108]define a complex query as a query consisting of one or more subqueries that are combined by conjunctions or disjunctions in an arbitrary manner. In their work on the Active Query Forwarding mechanism (ACQUIRE), they promote the use of these 'nested queries' and describe a mechanism that seeks to resolve the query in-network, generating information as a response. The work, however, does not address the implementation of the mechanism and instead presents a mathematical model that is used to analyse the performance of the approach in terms of energy cost.

[25]also identify the need for nested queries (which they too call complex queries) and highlight the problems with evaluating such queries especially in cases where aggregation dependencies exist between the nested queries. They put forward the idea of the query itself supporting abstractions which can then be used in query resolution. In their example the abstractions are geographical regions. Their research resulted in a qualitative study of the requirements of such a system and did not extend to implementation.

This research is therefore concerned with the processing of complex queries and first defines a complex query as one which:

1. consists of one or more subqueries (nested queries) and/or

2. contains multiple operations such as aggregates and/or.

3. contains spatial and/or temporal elements.

The query-based systems currently in use neither provide the facility to construct these complex queries nor give users the ability to process them. Such queries, for instance, queries with dependencies (nested queries) would require more complex in-network interactions than those supported by current query-based systems.

One example of a complex query which forms the object of the research here would be:

'What is the average temperature in those areas in the network where the humidity is greater than 95 and the air pressure is between 900 and 1000 mbar'.

This query exhibits a number of complex elements. First, the query language would have to be able to accommodate the expression of the spatial elements described as 'areas' in the example above. Second, the query is a dependent or nested query and can only be answered after some reasoning within and between the defined spatial entities. In effect, parts of the query depend on a previous question being answered and only become relevant if a particular answer is obtained.

With the primary goal of creating a system that exhibits simplicity and usability, using a declarative language already familiar to users of existing applicative query mechanisms as well as traditional database systems is considered a worthwhile approach. Although not examined in this report, the PhD program will identify and investigate existing SQL-based query languages in creating a language capable of expressing the complex query requirements described above.

1.2.2 In-Network Complex Query Processing

As will be shown, a number of in-network processing techniques have already been proposed in the literature and used in existing information extraction systems. These include techniques like aggregation, fusion and filtering which have been shown to improve energy efficiency in WSN systems and have been incorporated extensively in both query-based and agent-based systems. In addition, a powerful feature of many of the macroprogramming systems has been the creation and use of node or network level logical abstractions to facilitate in-network information processing. The literature has shown that so far logical abstractions have not been considered for application in query processing systems. The author believes that these logical constructs can provide a more powerful means for processing the complex queries identified as being of interest in this research work. The work examines the usefulness of abstractions that can be constructed logically within the network and used in conjunction with the in-network processing techniques identified above. The abstraction can be based on two types of attributes. Static attributes which do not vary over time, for example, the type of reading a node provides and dynamic attributes which do vary over time, for example, the current sensor reading. The key idea here is that the abstractions would be a component of the query itself and constructed prior to the dependent query being posed. These abstractions will drive the manner in which the query is both disseminated and processed within the network.

Consider an example where a user is interested in monitoring the soil acidity and relative humidity in 'hot patches' of the vineyard. The query posed would first need to define what the 'regions of interest' are. In this case, regions would be logically constructed over areas where the temperature level registers above a given threshold. Once these regions have been defined, the body of the query, aimed at retrieving soil acidity and humidity readings, will be disseminated to the relevant nodes via the region construct and not to any node within broadcast range, for instance. The idea is that the regions are queried rather than individual nodes.

Hence, the goal of the research is the creation of a novel query processing system that allows the construction and posing to the WSN of higher level requests (called complex queries), queries so far not addressed in existing query processing systems. The system will have as a key feature, query-dependent logical abstractions which will be used to facilitate query dissemination and processing in the network.

1.3 Methodology

The research is approached by first identifying a real-life application where in-network complex query processing is essential. It is anticipated that the identification of a motivating application will both inform the creation of realistic complex queries and also help establish a coherent set of application scenarios within which proposed processing techniques can be tested. The query types identified will form the basis for the investigative work into devising strategies, techniques and finally a framework for in-network processing.

Existing components and techniques which could contribute to the goal will be investigated and if possible utilized in developing the query processing system. The implementation of the system will be evaluated

both in simulation and on the Cogent testbed (a 40 node WSN). A number of performance metrics will be used in assessing the success of the system. These will include energy efficiency (for example, comparing power consumption during query processing), robustness of the approach to node or link failure, latency in getting query results to the user and scalability of the approach.

1.4 Report Structure

To set the context of the work presented in this report, it is important to first examine the current literature for approaches that have been used in information extraction in WSNs. Chapter 2 therefore examines these methods as well as a number of WSN architectures (which influence the selection of an appropriate information extraction method) and a number of routing protocols (which have an impact on information processing). Chapter 3 consists of two main sections. The first section positions the work in the thesis by describing the class of WSN applications the research targets and identifies a key motivating application that will be used as a basis for development of the prototype system. The second section describes the constraints and assumptions under which the work is carried out and outlines the system architecture. For the purposes of this report, some suitable in-network processing strategies from the literature were identified and tested in simulation. Appropriate corresponding information extraction techniques were developed by the author and were further tested and critically evaluated in terms of effectiveness and efficiency alongside existing centralized approaches. Chapter 4 presents these achievements and sets the stage for future work.

1.5 Contribution

A number of contributions are reported in this progress report. First, a comprehensive literature review has been produced which considered over one hundred and twenty five publications. The second contribution is the description of a novel complex query processing system architecture. Thirdly, the report also identifies candidate WSN architectures for which the proposed system is suited and discusses preliminary experiments which investigate the use of logical abstractions for complex query resolution. Future work will investigate the formulation of a suitable query language for composition of complex queries and the development of a distributed processing system for complex query resolution.

Chapter 2

Information Extraction Methods

By and large, the work in this research project is centered round information extraction in WSNs with particular focus on approaches that utilize in-network processing for generation of information. In general, there are three main approaches to information extraction from WSNs:

Agent-based: this approach tailors the information extraction mechanism to:

1. the type of information needed - for example, an agent may incorporate code for data aggregation as required by the application.

2. the configuration of the network the information needs to be extracted from - for example, a mobile agent's route through the network would be determined by the network topology.

An agent is simply a piece of software that performs a task without the need of user invocation to function. Agents are used to perform tasks, make decisions and collaborate to achieve more complicated tasks. Some applications are based on multi agent systems in which multiple agents sometimes programmed to function in different ways, collaborate, coordinate and organize to perform complex tasks and generate information.

Query-based: this approach takes the view that the user should be shielded from the internal workings of the information extraction method. With a focus on usability, query-based approaches promote the idea that the user should not need to have detailed knowledge of the underlying network configuration or how the mechanism works in order to retrieve information from and communicate with the network. Requests are initiated via queries written in an appropriate declarative language and information returned as a result.

Macroprogramming: this approach focuses on programming the network as a whole by directly specifying global behaviour instead of programming individual nodes. It involves the definition of high level languages that can capture operations going on in the network at a global level rather than at node level. Global behaviour is specified, programmed and then translated into code which runs on the nodes. Ideally the programmer is not concerned with low level details like network topology, radio communication or energy capacity.

This chapter discusses these three approaches to information extraction, identifies example systems, and highlights those that have been used in practical deployments. Wherever possible the architecture and mode of information processing (whether centralised or distributed) is identified. The key issues for discussion lay with:

1. identifying which of the methods proposed in the literature have been evaluated at implementation level. (This analysis is needed as a large proportion of the research effort is at theoretical level and not readily suitable for practical deployments.)

2. identifying which of the methods proposed in the literature make use of or support in-network information extraction. (This is important as approaches that utilise in-network processing like aggregation and filtering have been shown to be more energy efficient and therefore more desirable.)

2.1 Agent-based Approaches

The agent-based approach to information extraction tasks a network by injecting into it a program with some type of processing or decision making function. In this approach there is no attempt to devise ways of transmitting data to 'collection' or 'aggregation' points for processing, rather, the application is sent *to* the data instead [23]. The agent is able to collect local data and perform any necessary data aggregation. Autonomous agents can make decisions without user input. In the case of an autonomous agent, which also happens to be mobile, that decision may involve whether the agent should move to a particular node, and if yes, which node it should migrate to. Autonomous mobile agents moving from one node to another can be designed to be cognizant of issues that may exist in the network [129]. For instance, they can make decisions, perhaps to clone themselves, if necessary, to avoid failed regions or nodes in the network. If an agent detects that a node has a problem and is unreachable it can adapt its route so that it avoids the failed node [76]. The mobile agent therefore presents added flexibility in terms of decision making and reliability in dealing with faults in the network.

2.1.1 Agent-based Architectures

[121]report three architecture models for agent-based wireless sensor networks. The classification is based on the number and type of agents being used and the mode of operation of the agent-based system.

In the **first model**, all nodes respond to a single agent usually housed at the base station or central processor. The advantage is that there is a single control point with the benefit being the ability to access the entire network and an increase in efficiency since transmissions from this central point could be better synchronized and collision reduced as a result. Collisions during transmission require a retransmission which is quite costly. Therefore, synchronization brings with it direct energy savings. There are, however, a number of limitations to this model.

The first limitation is lack of scalability. In this model, the agent queries each node for a sensed value which forms the input to the agent's deliberation (the agent follows a perceive-deliberate-act cycle). As

the number of nodes increases, however, the time taken to deliberate also increases which can lead to time delays. Also, with larger networks it is more likely that multihop communication is needed. This can lead to even more time delays during data gathering as well as increased power consumption.

A second problem is observed in cases where there is a need for more than one agent to access nodes in the network simultaneously, for example, to execute different application tasks. Here an increase in the time delay occurs because one agent would have to wait for another to complete its task before accessing that node or set of nodes. This leads to delays in agents getting the data needed for processing and acting if needed. This model can be considered to utilize a centralized processing approach as all data is transmitted to a central point (albeit an agent) for further processing. In-network processing is absent.

In the **second model**, each node in the network hosts an agent which will determine how that node will behave. Control of the network is therefore distributed and agents can coordinate and collaborate to achieve goals both at a local and global level. A main advantage in using this model is its scalability. The ability to perform local computation without needing multihop messages to a central controller results in a reduction in energy usage. A second advantage is the ability of the network to perform tasks concurrently given the presence of multiple agents. However, in practice, it is difficult to create agent-based systems that solve global tasks using only local information. The result, therefore, is an increase in resource and energy usage for the necessary inter-agent communications, a cost avoided by the centralized model presented above. This multi-agent approach clearly incorporates in-network processing techniques for information generation. The "environmental nervous system" [74]a multi-agent system, is based on this model. It is a small (three node) autonomic wireless sensor network (see figure 2.2b) aimed at environmental sensing for intruder detection. This system is described in greater detail in Section 2.1.3.

The **third model** uses mobile agents. Here an agent is able to migrate to a node or nodes, collecting, aggregating and fusing data before it sends a message back to the central processor. A primary advantage here is a reduction in the cost of computation since only one node is active at a time. A second advantage is that communication cost is low since migration of agents only involves one *transmit* and *receive* event. The agent in this model is responsible for both collecting data and the itinerary related to the task in operation. The main disadvantage, however, is that if the node to which an agent has migrated fails then all data gathered up to that point is lost. There are, however, techniques that could be implemented to combat this problem, for instance leaving copies of the data on previously visited nodes so that the agent could restore its state if needed. A second disadvantage is the lack of concurrency this approach brings. In large networks, in particular, this could be a problem if the agent takes too long to collect all the data required to perform the given task. The mobile agent-based deployment reported by [98]is based on this model. This network was aimed at ground vehicle classification using acoustic sensors and is described in greater detail in Section 2.1.2.

Each of the three models described exhibits both advantages and disadvantages. According to [121]the strengths of the three methods were simplicity in the case of the centralized approach, robustness in the case of the multi-agent approach, and efficiency in the case of the mobile agent approach. The weaknesses were identified as scalability, complexity and latency. Two hybrid approaches were proposed by [121]that combined the features of the three in ways that attempted to minimize the impact of the disadvantages described.

The first hybrid model combines features of the first and second models and divides the network into

groups of nodes with an agent assigned to and responsible for all nodes in a group. Coordination between groups occurs via their assigned agents. The scalability issue is addressed here if the number of nodes an agent is responsible for is sufficiently small and the problem of inter-agent coordination costs (which was apparent in the second model where an agent was assigned to each node) is ameliorated since we have fewer agents in the system.

The second hybrid approach combines the second and third models and also divides the network into small groups of nodes. The difference here is that the agent assigned to a group is a mobile agent that migrates between nodes within a group. As in the first hybrid approach scalability is addressed and latency is reduced if the number of nodes in a group is small as well as a reduction in coordination costs is achieved because of the smaller number of agents. An additional advantage given the mobility of the agent is further reduction in the cost of communication (reduced to the cost of one transmission and reception). Much of the reported research in the area of agent-based information extraction mechanisms for WSNs describes variations of the agent approach outlined above. In the following, some examples of mobile and non-mobile agent system architectures will be examined in more detail. The section will describe models or mechanisms applied in utilizing both mobile and non-mobile agents for data collection in WSNs.

2.1.2 Mobile Agent-based Distributed Sensor Networks (MADSN)

The distributed sensor network (DSN) model was first proposed by [126]. In this model, referred to as a *fusion architecture*, all sensor data is sent to a central location where it is fused [129]. A number of variations followed [58, 95, 96], each making some improvement on the DSN architecture but holding to a common client/server paradigm or a centralized approach to information processing.

In a WSN with a large number of deployed sensors, each with its own piece of data, it is impractical to have all that information flowing to a central store. First, it has been identified that communication dominates power usage in sensor networks and that the cost of transmission of data from individual nodes in a network to a base station or sink far exceeds the cost of in-network processing of data before transmission [137, 115, 94]. With resource constraints like limited battery power on the nodes in the network, there is a need to reduce the cost of communication if at all possible. This problem is further exacerbated by the fact that often not all data that is transmitted is critical to ensuring quality information. There is often a large quantity of redundant or erroneous data. Other problems include network bandwidth limitations, unreliable connectivity and noise in the network. Given these constraints, DSN and other client/server type architectures cannot meet all the challenges of WSNs. Qi, Iyengar and Chakrabarty proposed an improved DSN architecture that used mobile agents (MADSN) [97, 98]. Figure 2.1 shows a comparison between DSN and MADSN.

Whereas in DSN sensor nodes collected data and transmitted to the base station node or sink (the central processor), with MADSN the computation code is transmitted to the node where the data resides. The need to transmit masses of data is removed and replaced by transmission of only a small piece of code representing the agent. Additional benefits are that the agents can be programmed to perform fusion tasks to consolidate data increasing the extensibility of the system and the mode of information collection is more stable since it is not affected by fluctuations in connectivity. If a network connection fails, the agent
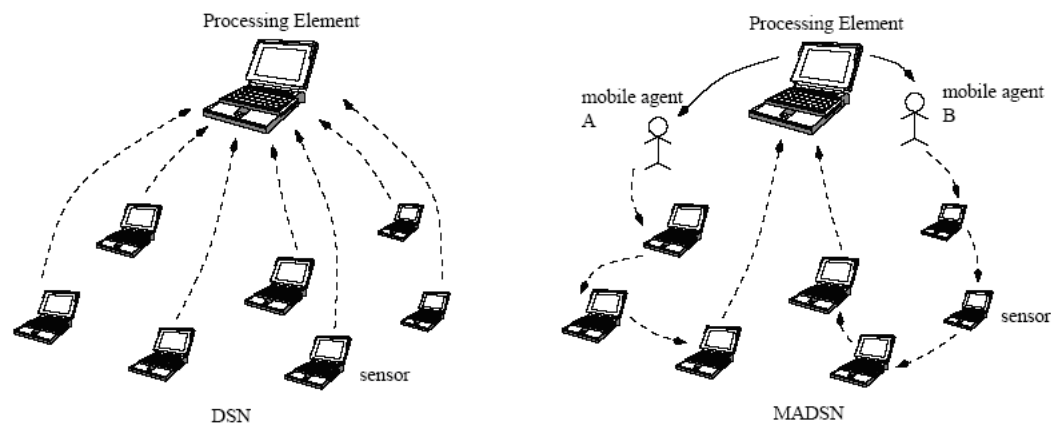
Figure 2.1: Comparison between DSN and MADSN architectures taken from [97]

can wait till it is re-established before returning its results.

An agent in MADSN is defined as an entity consisting of five attributes:

Identification - a 2-tuple representing the dispatcher's identification number and serial number

Itinerary - migration route information assigned by the dispatcher or processor

Data - the agent's data buffer carrying results of data integration

Method - implementation of algorithms (multi-resolution data integration algorithm)

Interface - provides interface function to enable communication between the agent and the processor as well as to give the processor access to the agent's data buffer.

The agent will selectively visit nodes in the network and fuse appropriate data. When it gets to the selected node it will present its credentials and obtain access to the node's local data or services [129]. The agent may collect the data or perform functions before leaving with the results.

A comparison between MADSN and DSN showed a 90% savings in data transfer for MADSN over DSN because of savings made in avoiding transfer of raw data. In some cases though, this was despite the overhead introduced by having agents created and dispatched [97]. In another study a model that incorporated collaborative signal and information processing (CSIP) techniques was applied and an analysis of performance between MADSN and DSN conducted to measure energy consumption and execution time. Again, in that study MADSN outperformed DSN [101].

The benefits of the MADSN approach can therefore be summarised as follows:

1. It requires less network bandwidth since only the agent is transmitted as opposed to large quantities of data.

2. It is more scalable since performance does not depend on the number of nodes in the network.

3. It is extendable since agents can be adapted to perform different tasks within the network.

4. If a connection were to go down the mobile agent could transmit results when it was re-established. Therefore performance is not greatly affected by the reliability of the system.

[99]studied the use of mobile agents for data fusion in a WSN running MADSN and focused on creating an optimum design of the *itinerary* component of the agent to improve performance and minimize resource usage. [100]showed that MADSN could be successfully applied to the real-world problem of vehicle classification in an unattended ground sensor system. The study showed, however, that classification accuracy was low (about 23%) when only one sensor was used but improved to about 80% when a multisensor array was used. Target classification accuracy with a single sensor was high (about 75%) only when the target was in close proximity to the sensor.

2.1.3 Autonomic Wireless Sensor Networks (AWSN)

[74]introduced the concept of the autonomic wireless sensor network. They proposed that mobile agents could be deployed in the network to facilitate autonomic computing. Autonomic computing means the ability to self-manage, self-maintain as well as interact with a user of the network at a policy rather than hardware level. The purpose of interaction would be to allow interoperability with legacy systems if needed. Mobile agents in that context would support cooperation and negotiation in the system via relevant protocols and a suitable agent communication language. The mobile agents would introduce flexibility in terms of the ability to modify agent code through agent migration or agent adaptation. The idea here is that an agent could evolve as system policies evolved. In modelling autonomic behaviour, an AWSN also uses the self-knowledge, self-protection and self-interest characteristics inherent in mobile agents of multi agent systems.

Autonomic behaviour in Marsh's system includes the ability to handle sensors which have been rendered inoperable due to some type of damage, and the system can put into effect strategies to deal with that. For instance, the authors suggest that the system could make an estimate of data lost due to system damage and effect nearby sensors to increase their sampling rate to compensate. It could also mean a reconfiguration of the routing topology to minimize or eliminate message loss. The AWSN concept is closely related to the multi agent system framework and incorporates some of the same ideas including that of entities migrating from node to node while making 'intelligent' decisions. Figure 2.2 shows the setup of the experiment.

[91]and [122], building on this idea of an autonomic wireless sensor network, proposed an agent-based approach to implementing intelligent power management. In their work they promote the use of agents in intelligently activating or deactivating nodes for power conservation based on interpolation. For example, the following criteria are used to decide whether a node should be put to sleep: a node is considered redundant if the remaining nodes can interpolate the temperature at its location to a desired degree of accuracy. Redundancy in this context means that the node is not needed since the other nodes or the network can operate effectively without it.
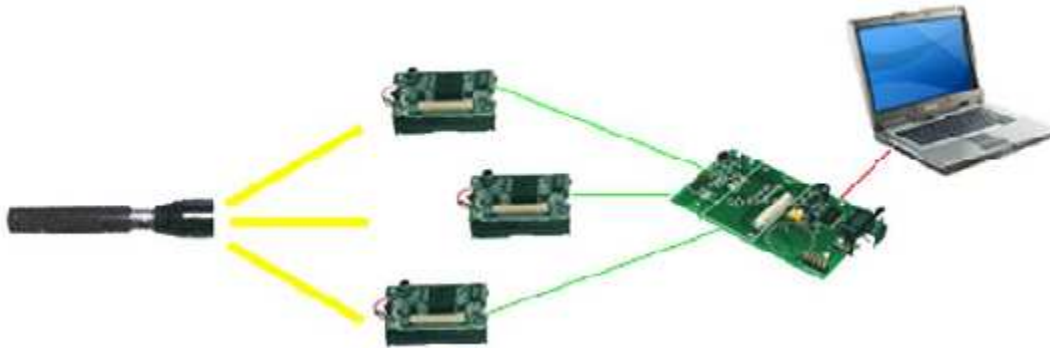
Figure 2.2: Setup of the intruder detector experiment, each node houses an agent, taken from [74]

In the authors' scenario, a *coordinator* agent is appointed and aggregates all calculations performed by other agents in deciding whether deactivation should occur or not. The coordinator agent sends out requests to other agents to calculate the components of the interpolation function used and send that result back to the coordinator. The coordinator then calculates the actual interpolated temperature. The coordinator then requests the temperature from the appropriate agent and if the two values fall within a given error range the node is put into sleep mode.

It must be noted, however, that the issue of power management is influenced by a number of other network characteristics such as coverage, latency and longevity. Any power management scheme would have to balance the tradeoffs in making decisions on how power can be conserved. In the above example, the inherent autonomic characteristics of the network itself are used for efficiently managing power usage. Although a promising approach, this concept has not yet been implemented in agent-based WSN deployments.

2.1.4   Mobile Agent-based Wireless Sensor Networks (MAWSN)

[23]identified that the operation of MADSN was really based on three main assumptions. First, that the network had a cluster-based architecture, second, that each source node (nodes with data) was one hop away from the clusterhead, and third, that the redundant data collected could all be fused into one packet of fixed size [23, 80]. Those restrictions in a real-world context seemed to impose too strict limitations on the applications that could make use of the architecture and excluded many systems which did not boast any or some of these features.

Identifying this as a problem, [23]proposed MAWSN as an alternative architecture to address networks where the features needed for the MADSN approach to work were not evident. Previous work done by [80]showed that in multi-hop environments without clusters, mobile agents could be employed to eliminate data redundancy by employing context-aware local processing techniques, use data aggregation to eliminate spatial redundancy with sensors that were of close proximity to each other, and reduce communication overhead by combining tasks. They were able to reduce information redundancy at three levels: the node level, the task level and the combined task level.

At the node level mobile agents were assigned processing code specifically targeting the requirements of the specific application. The result was that the mobile agent only needed local processing of that data requested by the application thereby reducing the amount of data transmitted as only relevant data was extracted from nodes for transmission. At the task level, the mobile agent aggregated sensed data from individual nodes when visited. Finally, at the combined task level, the mobile agent used the packet unification technique to unify shorter data packets to one longer packet again reducing the number of transmissions. MAWSN built on the encouraging results in [80]and experimentally showed improvements in energy consumption for data packets transmission over the more traditional client-server based WSN.

Concurrently with the above work, [119]proposed a framework aimed at implementing a wide variety of sensor network applications. Their aim was to create an architecture that allows multiple applications to share a network and permits scalable deployments that can evolve over time. In their design autonomous agents are deployed in the network with a particular agent having the ability to be on one or more nodes. Their framework is built atop TinyOS and its associated virtual machine Mate' [65]and agents run on Crossbow motes. The agents were shown to use resources only on those motes they visited so resources in the network were used more efficiently. Agents were able to read sensor values, start devices and sent radio packets if required.

In a data collection experiment aiming to find the maximum value in a field of sensor readings, an agent was injected into the network via broadcast. After arriving at a node, it re-broadcast itself if the reading at the current node was greater than that at the previous node. Agents stopped propagating once the region with the maximum sensor reading was reached. This was done by incrementing a count value when an agent arrived in the area. Once that value went over a pre-defined threshold the last agent to enter the area could issue a notification message to the processor.

Analysis of the experiment showed that as the size of the network increased, a smaller proportion of the network needed to be active. This meant that in large networks agents stayed on nodes only long enough to perform their task and quickly release resources for other task execution. The conclusion of the experiment was that the agent model presented was successful in showing that it could scale up without having to occupy all nodes on the network in performing tasks; however some areas for future work were identified. The first is security to ensure that unauthorized agents are not able to mount energy-depletion and denial-of-service attacks among others. The second involves giving agents the capability to query nodes for information such as processing power and energy. The third area involved giving agents the ability to cache their code on a node to reduce the number of agent transmissions.

In a similar approach [61]describe the agent oriented programming paradigm for development of intelligent sensor networks. They implemented a test application using the Java Agent Development Framework (JADE) [53]aiming to develop an *intelligent* ground sensor network. The application consisted of an Unattended Ground Sensor Network (UGSN) used to monitor moving targets with agents analyzing the data being acquired by nodes. In the experiment, information from the sensors were correlated and fused by reasoning agents using a fusing algorithm (the majority voting ordinary technique) suited for distributed information fusion. A special agent called the *Target Agent (TA)* is created on the node where the first sensor trigger occurs. The *TA* then reasons with its neighbours and after correlating their sensor data, decides what node it should migrate to next. The objective of migration is to get closer to the target being tracked. After each move the *TA* performs data fusion using the new data available on its host node. Through the *TA*, the network as a whole, decides when external applications need to be updated and

what information should be sent to those applications.

The experiment was successful in its sole aim of showing that agents were suitable for distributed information fusion. Distributed information fusion is the idea, quite similar to aggregation, that data generated by nodes in a network can be efficiently combined in-network instead of transmitting all raw data to a sink for processing. The main advantages of this approach are identified as:

1. reduction in data redundancy since we have agents deciding whether data is important enough to be used.

2. reduction in power consumption in having all data sent directly back to a central location.

3. conservation of communication bandwidth by limiting the exchange of data between nodes as much as possible.

The advantages of agent-based distributed information fusion parallel those exhibited by other systems, particularly query-based systems that incorporate in-network aggregation. Query-based systems are addressed in more detail in section 2.2.

2.1.5 Multi-Agent Systems

Although the idea of using agents in wireless sensor networks is widely supported by the literature [96, 101, 124, 99, 80] , the actual use of multi-agent systems in WSNs is not as widespread as one might expect [123]. There are of course the big problems of deployment, testing and debugging of such systems on what is essentially a distributed application with minimal interfaces to support user interaction. There have been real deployments using agents [129, 119, 106, 80]but multi-agent system deployments are still rare. [123]propose a methodology for multi-agent deployment. They start with a base station implementation for the agent which leads to a distributed implementation where agents are mapped to nodes, using either one-to-one, many-to-one or one-to-many mapping. This mapping is used to model the interaction expected between agents and the base station, as well as iron out communication and interaction rules between agents. These first two stages are carried out on a device with more processing power like a laptop or desktop. In the third and final stage, the statements and rules governing agent behaviour are translated to the language of the WSN system hardware that will be hosting the agent. The result of the last stage is a topology in which the load of executing an algorithm is distributed among the agents which can allow faster response times for complex algorithms. In addition, it can also result in a reduction in the number of transmissions in the network since packets which previously had to be routed to the base station for processing can now be processed by an agent on a node or on a neighbouring node.

2.1.6 Wireless Sensor Nodes as Intelligent Agents

In the examples described above, agents are pieces of code injected into the system, gathering data, making decisions and migrating between nodes in some cases. Some research, however, has investigated the concept of having nodes themselves act as intelligent agents. [79]use intelligent agent theory

to model sensor behaviour, viewing the WSN as a network of distributed autonomous nodes with the capability of making decisions. They also propose using artificial intelligence strategies on nodes to enable decision making, selecting a rule-based expert system that makes use of locally collected information on a node to make step-wise decisions. There is also a proposal to incorporate some fault-detection strategies so that nodes can detect failing neighbouring nodes and make routing decisions using that information.

The work also selected the directed diffusion routing protocol [54]since it presented a localized, data-centric protocol endowing each node with the same routing, sensing and aggregation capabilities. In addition, directed diffusion promotes a view where nodes use local data to make decisions; a view mirrored by [79] in the approach taken in their work.

2.1.7 Agent-based Middleware

In general terms, middleware sits between the operating system and the application with its main function being to provide support for development, deployment, execution and maintenance of sensing applications [105]. There are a number of categories of middleware based on the objectives of the approach and the way in which the wireless sensor network is viewed. With some, the main goal is to provide a dynamic reprogramming capability while others seek to provide a platform-independent model on which sensor network programs can be written and executed. Other approaches try to provide a cross-layer management approach and provide functionality for manipulating other services like routing, etc. Information extraction which is ultimately the goal of the sensor network applications built using these models is supported by the services provided by the particular middleware approach. Approaches include the use of virtual machines which in many cases are application-specific but reduce the amount of code that has to be transmitted and therefore reduces the cost of communication, as well as modular programming approaches which allow easier and more efficient reprogramming of sensor nodes [117]. Some agent-based middleware will be described below.

Mate

Virtual machines (VMs) provide one middleware solution to the problems posed by wireless sensor networks. In WSNs the greatest drain on energy resources is the cost of communication, so if the traffic needed to reprogram motes could be reduced this could lead to a longer network lifetime as well as the ability to reprogram more frequently. Mate is a middleware approach that abstracts high-level operations into virtual machine bytecodes (see figure 2.3). As a result a VM program can be very short, bytes long instead of kilobytes. Mate, therefore, is a virtual machine designed for sensor networks [65]and forms the basis for many agent-based information gathering applications. For this reason it is evaluated here.

Mate is a bytecode interpreter, running on TinyOS and provides a high-level interface for creating small (<100bytes) but complex programs. Mate is focused on energy conservation in transmission of code and breaks code into small sections called *capsules* (referred to as agents by some), each 24 bytes long. Programmers write applications and the system injects them into the network using certain algorithms that minimize energy and resource usage. The small size of the modules makes it easier to distribute into the network. Mate specifically targets the constraints of limited bandwidth and power by allowing
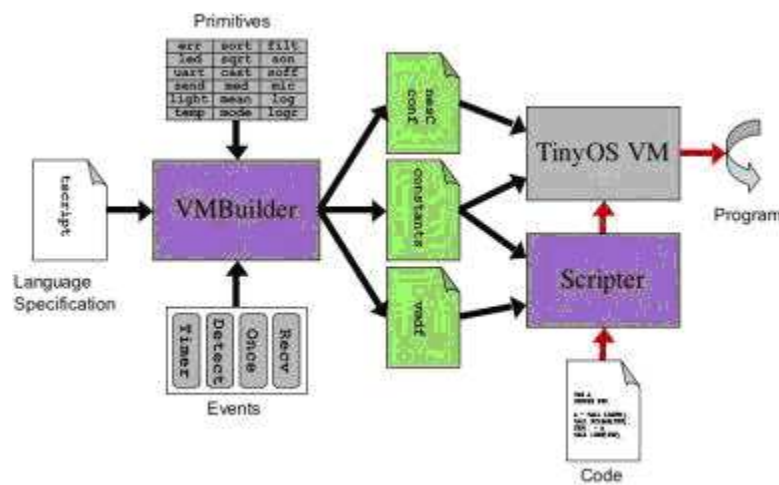
Figure 2.3: Construction of a Mate virtual machine in compiling and running a generated program taken from [77]

adjustments or reprogramming of capsules before they are issued to the network.

Mate is a general architecture that allows a user to create a wide variety of virtual machines. A user builds a Mate VM in three steps [105] :

1. The user selects a language that defines a set of VM bytecodes representing the basic functionality.

2. The user selects the execution events. Each event has its own execution content which runs when an event is triggered.

3. The user selects the primitives to be used. These are operations that provide functionality beyond that of the selected language.

A set of files are generated after these steps have been executed. Some will build the VM which will run on the motes, others a scripter program with information on the language and primitives selected. The user can now construct programs in the scripter which will be compiled down to the VM-specific binary code. These bytecodes are then issued to the network and processed by a VM there which will execute it.

To reprogram motes, the user needs only add one mote with the new capsule to the the network. When a mote hears a new capsule it immediately starts forwarding it until every mote in the network has been updated.

The advantages of Mate, therefore, are the simplicity of programming for the sensor network as well as the ease with which the network can be reprogrammed. The authors have identified future work as being the creation of propagation algorithms for larger programs, security and scripting languages that can be compiled to a Mate VM.
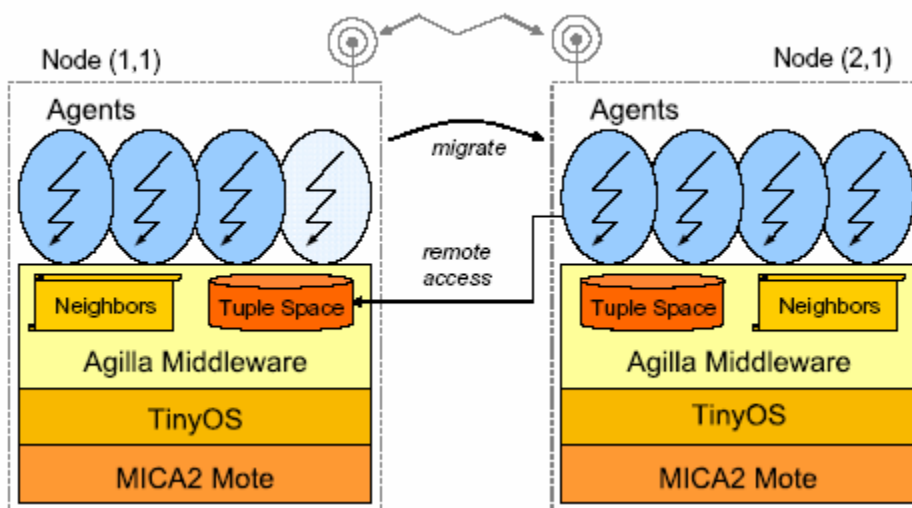
Figure 2.4: The Agilla programming model taken from [34]

Agilla

Agilla is a mobile agent middleware based on Mate that allows quick deployment of adaptive applications [33, 34]. Instead of creating capsules which are flooded throughout the network, Agilla allows the user to create mobile agents which can be injected into the network. Agents are dynamic and intelligent, and can coordinate and collaborate locally or through remote invocation. These agents can clone themselves and move from node to node as they perform tasks. Agilla affords more flexibility because it allows applications to decide how agents in the system should migrate and spread in the network while maintaining their code and state.

The Agilla Model

Each node can have a maximum of four agents and addressing is done using the geographic location of nodes instead of IDs. Agilla can therefore be used in running applications over geographic regions and to allow geographic routing for any multi-hop interactions [33]. Inter-agent coordination in Agilla is facilitated using a tuple space and an acquaintance list which are maintained on each node. Each node's tuple space is shared by local agents and remotely accessible. Global tuple spaces are not supported due to the energy and bandwidth constraints, rather separate local tuple spaces are maintained by each node and agents can access remote tuple spaces using special instructions provided in the middleware. Each node also maintains an acquaintance list containing the location of all one-hop neighbours and local agents can access it via a number of provided functions. Figure 2.4 shows the programming model for the Agilla system.

[34]used Agilla to successfully deploy fire and tracker agents in a fire tracking application and demonstrated the reliability and efficiency of the exercise in this case study [33]

The Fire Tracking Application

Background: a fire starts in a region of the network and as it spreads *tracker* agents crowd round it repeatedly cloning themselves until a perimeter is formed. Once that perimeter is formed a fire fighter is informed. The fire fighter injects a *guidance* agent who is responsible for guiding the fire fighter along a safe path to the fire. The study focused on the *tracker* agent, leaving the *guidance* agent and the development of a *safe-route* algorithm to future work.

Fire Agents

Two types of fire modelling agents were used, static fire agents and dynamic fire agents. Static agents simply insert a fire tuple into the local tuple space and then repeatedly blink a red LED (visual indicator of network state). These agents are used to create fires of different shapes. The dynamic agent models a spreading fire. It inserts a fire tuple upon arrival at a node, blinks the red LED a number of times, clones itself onto a random non-burning neighbour and repeats the blinking. The cloning and blinking process is repeated until every node is on fire. The rate of spread can be controlled by the number of times a node blinks between cloning operations.

Tracking Agents

A fire tracking agent is responsible for discovering a fire and forming a boundary around it. The tracker agent inserts a tracker tuple when it arrives at a node. Other tracker agents will use this tuple to check whether a neighbouring tracker agent is still present on the node. If the node a tracker agent is on catches on fire the tracker agent dies. A reaction is registered when this occurs, the tracker agent will turn off all LEDs, remove its tuple and stop functioning.

The life cycle of a tracker agent is as follows:

It repeatedly checks whether any of its neighbours are on fire. If there are none, it moves to a random neighbour and repeats the check. If, however, a neighbour is on fire it enters a tracking mode and lights up its green LED and repeats the following. It determines the locations of all neighbours that are on fire and for each non-burning neighbour within a certain distance of the fire clones itself to those nodes. This process is repeated until the fire dies. Periodically checking for neighbours close to the fire allows the agent to adjust the perimeter. Once a fire was started, a tracker agent was injected next to it. This allowed the investigators to focus solely on the efficiency of perimeter formation and not on fire discovery as well.

The experiment was successful in a number of ways. First, it demonstrated the use of Agilla in deploying complex applications. Second, it showed that multiple applications could share the network at the same time (fire-simulation and tracker application). Third, it showed that mobile agents can be used to successfully program WSNs. Experiments on a 26-node MICA2 network showed that tracker agents each 101 bytes long were able to form a perimeter around a static as well as a dynamic fire. It was also evident that the efficiency of perimeter formation, in the case of static fires, depended to a great extent on the amount of agent parallelism in the system. The authors have identified areas of future work aimed at allowing new instructions to be added to the network after deployment and also allowing the instruction set to be customized depending on the agent being deployed. [33]describes the experiment and its results in greater detail.
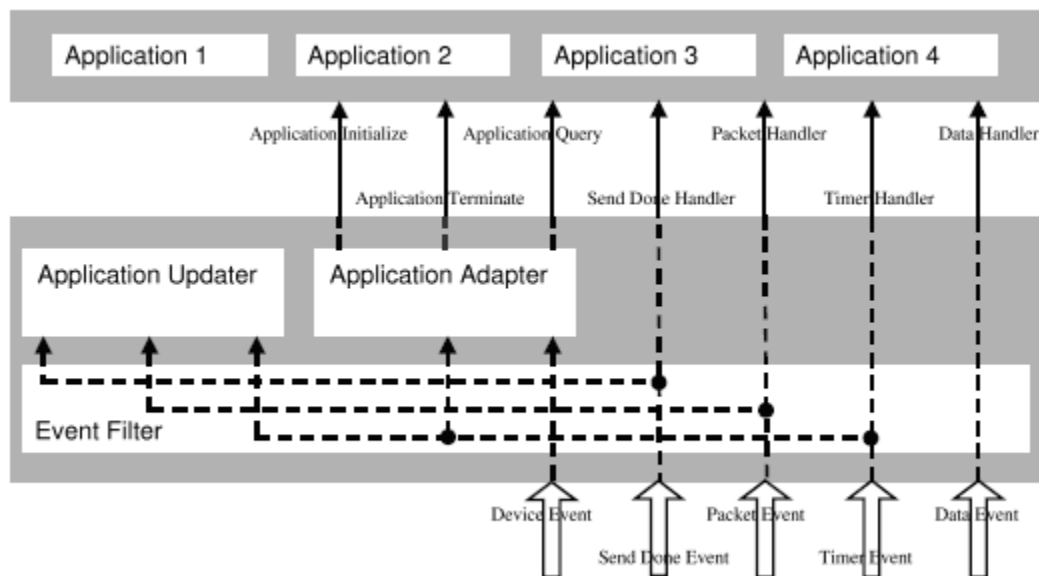
Figure 2.5: Architecture of Impala system taken from [120]

Impala

Impala can be considered another mobile agent-based middleware approach. Its middleware architecture allows a user to create modular, adaptable and maintainable applications for WSNs. The adaptation facilities allow changes to be made at runtime in an effort to improve energy-efficiency, reliability and performance of running applications. The idea is that given the need for long-term management of a sensor application, a middleware layer that can update and adapt applications dynamically, switch to new protocols easily at runtime is a big advantage. Impala, therefore, is a middleware layer that acts as an operating system, resource manager and event filter upon which applications can be installed and executed [120]. It was specifically designed for a wildlife monitoring project, ZebraNet, also detailed in [120].

System Architecture and Functionality

The system architecture is depicted in figure 2.5. The architecture's upper layer contains all application protocols and programs for ZebraNet. In the lower layer are three middleware agents: Application Updater (AU), Application Adapter (AA) and Event Filter (EF). The AU receives and transmits software updates to the node via the wireless transceiver, the AA adapts the protocols to different conditions at runtime in a bid to improve energy-efficiency and performance and the EF collects and sends events to the system for processing.

A program module is compiled into binary code before it is injected into the network. Linking is performed by the updater on every node and when all modules in an update have been received the module is linked to the main program. Modules are linked independently. Agents work autonomously making the network more fault-tolerant and better able to self-organize. It is not suitable, however, for resource-limited systems.

Impala uses an event-based programming model where the AA, AU and applications are programmed as a set of event handlers called by the EF when appropriate events occur. The EF controls various operations and starts a number of processing chains including timer, send, done data and device events. Based on different scenarios such as improving energy efficiency or performance, the AA will adapt a particular application. The AU is then responsible for effecting software updates while being cognizant of such issues as resource or bandwidth constraints and node mobility. If, for example, the radio transceiver on a node was to fail, the EF would call on an appropriate handler in the AA. The AA would determine the impact of that failure and decide whether an updated or new application that takes into account these issues should be issued to the network. The AU would then be responsible for carrying out the software update itself.

SensorWare

Although not strictly considered mobile agent architecture by some, SensorWare [15]is based on a scriptable lightweight run-time environment suited for nodes with energy and memory constraints. It is formally called an active sensor framework (ASF) where mobile scripts less than 180Kbytes are used to make the network programmable and open to users.

The difficulty in designing an ASF has been identified as determining how to accurately define the abstraction of the run-time environment. Any abstraction would have to result in compact code, resource sharing, multiple users being able to access the system and portability to different platforms. With these in mind they designed a node abstraction that allows multiple users access to the modules on a node while still being able to create new modules.

The authors focused on defining a framework that allowed the description and deployment of distributed algorithms for wireless ad-hoc sensor networks. SensorWare's language model can effectively express these algorithms while simultaneously shielding the user from low level details while allowing sharing of node resources among several concurrently running applications or many users. The language model focuses on the properties of efficient algorithms for sensor networks while application development for a real network is underway. Figure 2.6 shows the architecture of the SensorWare system.

SensorWare is also quite effective at dynamically deploying the distributed algorithms represented in the language. Given that the nodes are memory-constrained and cannot store every application in local memory the method of dynamic deployment has to be effective. Rather than each node being programmed by the user the SensorWare approach gets the nodes themselves to program their neighbours. The user injects the program into the network and the program is autonomously distributed to the nodes that should receive it. SensorWare is described in detail in [14].

TinyLIME

Unlike many of the examples described above that use a single controlling processor or sink node for data collection and processing, TinyLIME instead promotes a distributed approach. Multiple mobile clients or agents are distributed with the ability to receive data only from the sensors that they are directly connected to. The clients can share this locally collected data through their wireless interconnections [28]. TinyLIME
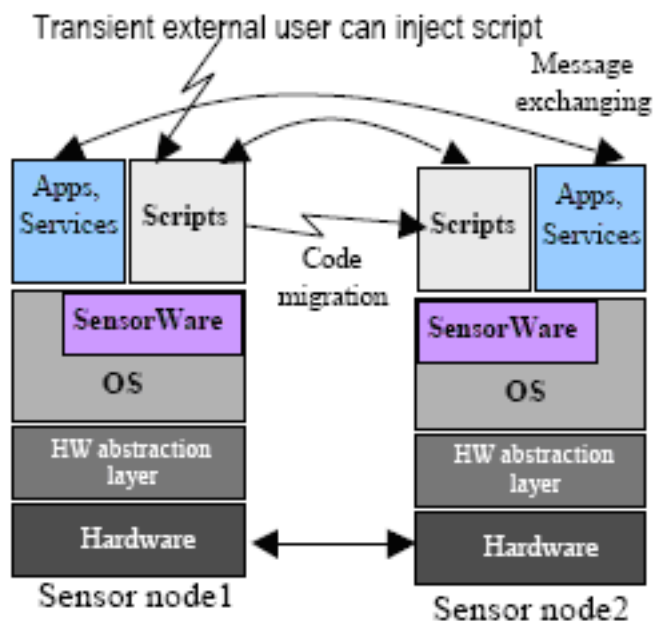
Figure 2.6: Architecture of sensor nodes running SensorWare taken from [15]

is built on the original Linda in a Mobile Environment (LIME) architecture [84]and deployed on Crossbow motes. All base stations run an instance of moteAgent, a LIME agent that among other functions maintains data freshness and historical information on recent sensed values. TinyLIME also supports data aggregation both over values sensed by multiple sensors and those sensed by a single sensor.

2.2 Query-based Approaches

Many researchers have taken the view that the sensor network can be considered a database from which information has to be requested and retrieved. The reasons for this are as follows: first, the network is a collection of data albeit that data is dispersed across multiple nodes. Second, data needs to be retrieved from the network, and like a conventional database one needs to be able to query the network for what is desired and get a response. Third, the nodes in the network are of interest primarily in as far as the data they generate, this can also be termed a *data-centric* view of the WSN.

Many of the *sensor network databases* use a query language similar to the Structured Query Language (SQL) for constructing queries.

SQL is the most popular language used for creating, modifying and retrieving information from relational databases. It consists of a number of clauses including `SELECT - FROM - WHERE` and `GROUPBY` clauses which allow selection, join, projection and aggregation respectively. SQL-type queries constructed for WSNs are quite similar to those of traditional SQL both syntactically and semantically. The FROM clause, for example, can be used to either specify sensors or data stored in tables. The `GROUPBY` clause allows a

`SELECT` statement to collect data across multiple records and group the results by one or more attribute values. In the WSN querying context, for example, records can be grouped by the node's id or by locations, etc. if these are attributes retrieved in the query.

In traditional database systems, data is accessed via an application or directly via a front end. Such applications insulate the user from the inner workings of the database system while allowing easy and meaningful queries to be applied to it in retrieving the necessary information. In a similar way, the sensor network can also be interfaced using a suitable application. The application would act as a query processor-type interface to the network taking into account the power and computation resource limitations on the devices in the network. According to [114]the challenge in query processing is not in processing data as quickly as possible but rather in figuring out a way to effectively respond to queries while transmitting as little data as possible. Several researchers have noted the benefits of this query-processor interface approach.

[12]describe a view of the network as a database with two methods for processing queries issued to the network: warehousing and distributed approaches. With the warehousing approach the processing of a query is separated from the interaction with the network itself. Essentially, the data is extracted and stored at a centralized location for processing. Queries in this case are pre-defined and usually ask for aggregate historical data, for example, "for each rainfall sensor, display the average level of rainfall for 1999" [11]. This model really mirrors a client/server approach to data collection as is found in a traditional distributed network [110].

There are two main disadvantages to this approach, however. First, it is not well suited to requests for continuous data, either because it is not possible to retrieve the required data from the node or because data is not retrieved frequently enough within the network to be able to answer a query [11]. Second is the high energy consumption that occurs when transmitting large quantities of data from the node to the centralized database and even more so in the case of a continuous stream of data, making the process very energy-inefficient [11, 12].

Alternatively, [11, 13, 12]propose a view of the network as a set of distributed databases where the composition of the query in effect determines what data is retrieved. Such queries have the advantages of efficiency because only what is required is actually retrieved, as well as flexibility since various queries can be formulated depending on what information is needed. This approach also takes advantage of the processing power on the node itself and where possible processes queries or parts of queries on them.

[40]note that data generation and routing in the sensor network can be seen as similar to the concepts of data storage and query processing in traditional databases and so also view the sensor network as a database. They examine the challenges in implementing the network as such and identify the need for robustness of data access given the possibility of node failure and noise that can affect readings. The idea is that by creating a standard querying interface that allows less restrictive query semantics, approximate results can be obtained which can help in producing an energy-efficient implementation of the 'sensornet database application'. [40]are also of the view that query optimization is closely linked to routing and they propose an adaptive approach that takes into account the volatile nature of data and communication in the network.
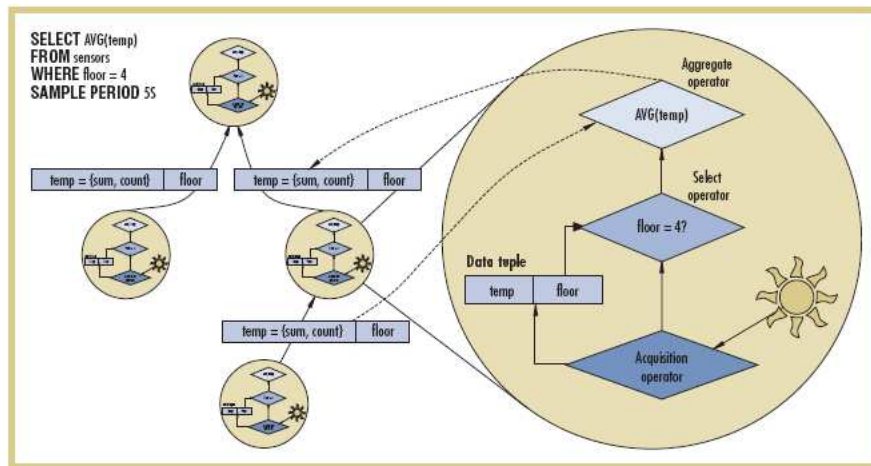
Figure 2.7: A sensor network running the Cougar system executes a simple aggregate query taken from [37]

2.2.1 COUGAR

The COUGAR approach has been proposed in [11, 13, 12, 132]and more fully described in [133]. It builds on the Cornell PREDATOR object-relational database system [111]and models each sensor in the network as an abstract data type while signal processing functions are modelled as abstract functions returning sensor data. The network is viewed as a system where each node is a 'mini database' holding part of the data contained by the whole. COUGAR allows the issuing of declarative queries for information requests and uses a query optimizer to plan an in-network processing strategy. Queries in Cougar are SQL-based and are posed to a virtual table called sensors with one row per node per instant in time and one column for every attribute. (The same model is used in TinyDB and is described in section 2.2.5).

A query proxy layer is placed on each node and interacts with both the routing and application layers in the system. A query optimizer is placed on a gateway node and distributes the query processing plans after it receives the queries from the user. COUGAR uses catalogue information and query specification to create the query plan which describes not only the flow of data between sensors but how information is to be computed on individual sensors. Once the plan is complete it is sent out to all relevant nodes. During execution, data is returned to the gateway node. In addition, a query proxy can inject queries into the network from arbitrary or specified nodes as it performs high-level services. Figure 2.7 shows how the system executes a simple aggregate query. COUGAR is especially useful for executing continuous queries.

A big advantage of COUGAR is that it shields the user from needing to have any knowledge of the underlying network, or how data is generated and processed. A second advantage is that it increases efficiency in terms of energy consumption since in-network processing is allowed thereby reducing the amount of data that has to be sent back to the gateway node [31]. There are other architectures which promote this distributed approach to query processing, for example, IrisNet, described in [85], SINA which is discussed in Section 2.2.2 and TinyDB which is detailed in Section 2.2.5 below. Both TinyDB and Cougar
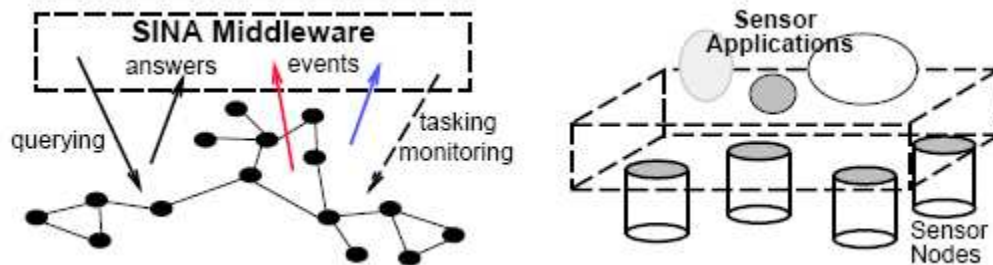
Figure 2.8: Model of a sensor network and SINA middleware taken from [24]

follow a very similar query processing model and address the resolution of both simple and aggregate queries. They both, however, **do not** facilitate the processing of other types of complex queries such as spatio-temporal or queries which target only a subset of the network.

2.2.2 Sensor Information Networking Architecture (SINA)

Like COUGAR, the SINA architecture described in [24]promotes a distributed database query interface that emphasizes in-network processing and reduction in power consumption in the network. It views the network as a collection of datasheets (a spreadsheet) with each datasheet containing the attributes relevant to the node it describes. A cell in this scheme represents an attribute and the collection of datasheets form what is called an associative spreadsheet or spreadsheet database representing the network. Each cell is therefore referred to using an attribute-based naming method.

SINA also makes use of clustering of low-level information in the network to increase efficiency. Clusters are formed from aggregations of sensors based on their power levels and proximity. Recursive aggregation can also be used to produce a 'hierarchy of clusters'. A cluster head within a cluster can then be elected to perform key functions like information filtering, fusion as well as aggregation [24]. Figure 2.8 shows a model of a network with the SINA middleware.

SINA uses the sensor programming language Sensor Query and Tasking Language (SQTL) which serves as the programming interface between applications and the middleware. SQTL is a procedural scripting language built from Structured Query Language (SQL) and messages written in the language can be interpreted and executed by any node in the network, although messages can target specific nodes if required. SQTL also supports the generation of information as a result of the occurrence of or change in some phenomena (events). Three types of events are supported by SQTL. The first is an event generated when a message is received by a node, the second is an event triggered periodically by a timer, and the third is an event triggered by the expiration of a timer.

SINA modules run on each node in the network and allow querying as well as monitoring of events. Nodes are aggregated to form clusters and elected cluster heads perform filtering, fusion and aggregation tasks. The user issues a query and the SINA architecture selects the most suitable methods for information gathering and distribution based on the type of query issued as well as the current status of the network.

A node receiving the query will interpret it and request information from neighbouring nodes in evaluating it.

A number of information gathering mechanisms are employed to help reduce resource consumption and increase response quality. These are: sampling operation, self-orchestrated operation and diffused computation operation, described in detail below.

1. **Sampling Operation** - for some applications a query may need to target the entire network in eliciting information of interest. Responses from all nodes in a network may result in response implosion [107], however, the effect can be reduced if nodes are able to make decisions on whether they should respond or not. SINA implements this 'decision-making' capability by assigning each node a *response probability* which determines whether they respond or not.

2. **Self-Orchestrated Operation** - in networks with a small number of nodes it is sometimes necessary that all nodes respond in order to improve the accuracy of a result. SINA uses what is called self-orchestration in an effort to reduce the problem of response implosion mentioned before. Here each node suspends its response transmission for a particular period of time to help reduce the likelihood of collisions occurring as could happen if nodes transmitted simultaneously.

3. **Diffused Computation Operation** - this method is used to implement aggregation functionality in the network. Each node is first assumed to have knowledge of its immediate neighbours and aggregation logic already programmed into the SQTL scripts is used to perform aggregation before routing the result to a designated node.

SINA is therefore particularly suited for aggregate and queries for replicated data in a cluster-based network configuration but does not address other complex queries like spatial or temporal queries which may also be suited for processing in a cluster-based configuration.

2.2.3   Data Service Middleware (DSWare)

Like SINA and Cougar, DSWare [113]is a data-centric middleware approach to providing information retrieval services within a WSN. DSWare, however, is aimed at handling real-time events and integrates a number of real-time data services while providing a database-like abstraction. In this way it can be considered another database-like approach adapted to sensor networks. It provides services for reliable data-centric storage and implementing alternative methods for improving real-time execution performance, as well as reliable data aggregaton and decreased communication which are aimed at conserving the limited resources available within the network. Its architecture separates the routing layer from the DSWare and network layers as DSWare provides components for improving power-awareness and real-time-awareness of routing protocols (see figure 2.9).

There are five services/components available in this middleware approach and these are shown in figure 2.10.

1. **Data Storage** - this component allows storage of location-linked data so that future queries for similar data would not mean having the query re-issued or flooded through the entire network. In
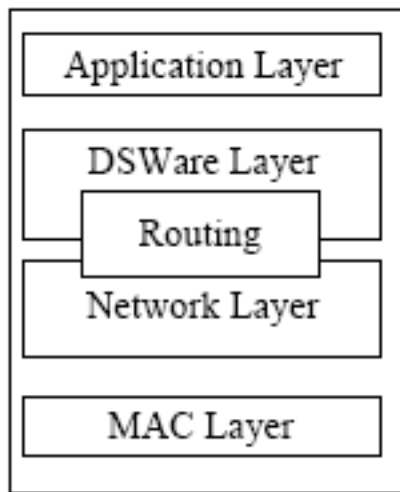
Figure 2.9: DSWare architecture, taken from [113].
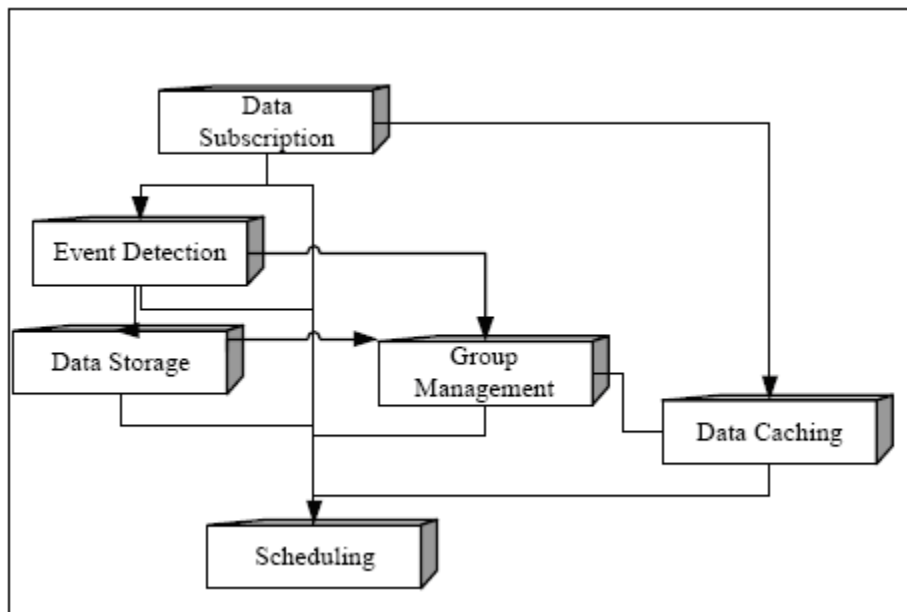
Figure 2.10: DSWare Framework taken from [113].

DSWare data lookup is implemented using a hashing function that maps data to physical storage nodes via a unique identifier while robustness is facilitated by using replicating needed data in several physical nodes which can then be mapped to a single logical node. Queries targeting that data can be directed to any one of these nodes in an effort to avoid collision and sustained overload on a single node.

2. **Data Caching** - this service provides copies of data that is requested often and spreads it through the network using the routing path. This serves to accelerate execution of the query while reducing communication aimed at accessing data for query processing.

3. **Group Management** - this component facilitates cooperation among nodes to accomplish more complex tasks. In some cases, for example, requested information requires multiple sensors combining their values to calculate a result. The group facility therefore can create a group when a query is issued. Nodes receiving some information on this group can decide whether they match the particular criteria described for that group. The group then manages the values of relevant sensors in accomplishing the given task after which the group is dissolved. In some cases, the query itself may expire before the group can be dissolved.

4. **Event Detection** - DSWare is based on event detection, an event being an activity that can be monitored or detected in the environment and is of interest to the application. An event is pre-registered depending on the application and can be classified as either atomic events (events that can be determined based on the observation made by sensor) or compound events (events than cannot be determined directly but rather need to be inferred from other atomic or 'subevents'). Like COUGAR, DSWare uses statements in a SQL-similar language for registering and cancelling events. After parsing the statement defining the event, DSWare generates the query execution plan and calls on the methods required to register, execute and cancel the event.

5. **Data Subscription** - this service is used to aid in more efficient data dissemination. If multiple nodes request the same data the subscription service puts copies at intermediate nodes which can then be accessed by the requestor nodes. This helps in reducing communication and helps conserve resources within the network.

6. **Scheduling** - this component provides a real-time scheduling mechanism as the default method with the option of adding on an energy-aware mechanism if the first has already been successfully implemented. All components of DSWare are scheduled using this component.

DSWare, because of its emphasis on providing event detection services and the mechanisms it provides for data caching, group management and data subscription, is particularly suited to aggregate queries as well as queries for replicated data. These aggregate queries, however, are simple aggregates over an attribute and do not include more complex aggregate queries where data from multiple clusters, for example, could be combined to resolve such queries.
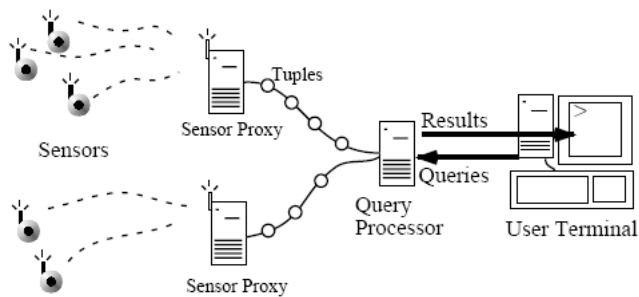
Figure 2.11: Fjords query environment taken from [67].

2.2.4 Framework in Java for Operators on Remote Data Streams (Fjords)

The Fjords architecture presented in [67]shows how processing multiple queries can be managed over multiple sensors while allowing more efficient resource usage and keeping query throughput high. The focus in the research is on creating the underlying architecture that will support the processing of multiple queries over sensor data streams.

The architecture comprises operators configured for streaming data and Fjord operators called sensor-proxies whose function is to serve as a mediator between the query processing plan and sensors. Data is allowed to flow into the Fjord from the sensor and then pushed into the query operator. Query operators are not active pulling in data but rather wait till data is sent to them from the sensors. The fjord, in addition to combining streaming and fixed data also processes multiple queries and combines them into a single plan. Figure 2.11 shows the query environment.

The Fjords architecture is therefore suited to continuous queries and addresses a number of issues important to query processing. It does this in two ways. First, by using proxies, non-blocking operators and query plans which allow streaming data to be pushed through operators that pull from data sources, it allows merging of the both stream data and local data. Second, by letting proxies serve as mediators between query plans and sensors it allows query processing while at the same time taking into account power, communication and processor restrictions in the network.

The work here is similar to the COUGAR project in that they both focus on processing of streams of sensor data, however, COUGAR although it does incorporate in-network processing in the form of data aggregation, it does not focus specifically on energy efficiency and the resource constraints on sensor devices but rather on modelling the streams of data using abstract types. Fjords, however, look more closely into improving efficiency for processing of data streams. Although a viable approach for continuous queries it does not address complex queries or even aggregate type queries to any great extent and is more concerned with managing simple query processing over multiple sensor data streams.

2.2.5 TinyDB

All the approaches to query processing described above have one thing in common: they view query processing in the network as a modified version of that in traditional databases. In essence, each node in the network is seen as a generator of named data against which queries can be issued. [71]describe a distributed database approach that attempts to improve energy consumption by applying varying techniques for aggregation and optimization within the network. This aggregation service is called Tiny Aggregation TAG) (section 3.3.3.1) and is designed specifically for TinyOS [50]motes. TinyDB has been one of the more widely used and popular query processing systems and considerable work has gone into extending its capabilities both in terms of the types of queries that can be issued as well as improving its operation by integrating different routing protocols. TinyDiffusion [109]is one such example. It has been widely used in a number of real-life deployments as well as inspired the research into other query processing systems and hence will be discussed in full in this section.

TinyDB uses acquisitional query processing (ACQP) as described in [72]for in-network query processing. In terms of practical applications ACQP focuses on the location and cost of accessing data as a way of reducing power consumption. There is no assumption made that data exists at a particular location rather queries are only issued where it is known that data exists.

TinyDB can therefore be considered an ACQP engine, a distributed query processor that runs on all nodes in the network. TinyDB was designed to be deployed on the Berkeley Mica mote platform which runs the TinyOS operating system (Mica motes are arguably one of the most popular of WSN hardware platforms.)

Query Syntax

Queries in TinyDB are quite similar to SQL with the FROM clause being used to either specify sensors or data stored in tables (called materialization points).

Sensor tuples are stored in a table (*sensors*) with one row per node per instant in time and one column representing each attribute. An attribute could be light, temperature, sound, etc. Records in this table are stored for a short period usually and only 'acquired' when needed to resolve a query. An example of a query could be:

```
SELECT nodeid , light

FROM sensors

SAMPLE PERIOD 1s FOR 10s
```

This query states that each node should return its own id and light reading from the *sensors* table once per second for ten seconds. The results are either returned to the user or logged. The sample period is used to indicate when data collection should begin.

The sensors table is an unbounded, continuous data stream of values and can only be sorted or used in joins via a specified *window*. TinyDB allows creation of windows via a CREATE statement where a storage point can be defined for example,

```
CREATE
```

```
STORAGE POINT recentlight SIZE 8

AS (SELECT nodeid, light FROM sensors

SAMPLE PERIOD 10s)
```

This statement specifies a node that will be used to store a stream of data. That data can be accessed by other queries and can be used in joins with other storage points on the node or joins with data in the *sensors* table. (In relational algebra a join allows one to establish links between data in different relations or tables by comparing attribute values within them).

Time Synchronization

TinyDB adopts the Timing-Sync protocol presented by [36]which attempts to provide network-wide time synchronization in the sensor network. The algorithm proceeds in two stages.

The first, the **Level Discovery Phase**, establishes a hierarchical topology for the network and occurs at the point of network deployment. The root node is assigned a level 0 and initiates discovery by broadcasting a *level_discovery* packet. This packet holds the identity and level of the sender. Neighbours receiving the packet, assign themselves a level which is one more than the level of the node from which they received the packet. These nodes then broadcast their own *level_discovery* packet to their neighbours. Once a node has been given a level it ignores any further packets it may receive as a way of minimizing congestion that occurs with flooding. The process continues until every node is assigned a level and results in a hierarchical structure with only one root node at level 0.

The second phase is the **Synchronization Phase**. Here a two way message is used to synchronize two nodes. Given two nodes A and B, A sends what is called a synchronization_pulse packet to B. This packet contains the level number of A and the value T1 which is the time measured by its local clock. B receives this packet at a time T2 which is equal to the time T1 plus the clock drift between the two nodes as well as the propagation delay time. B then sends back an acknowledgement package at T3 which contains the level number of B and the times T1, T2 and T3. A receives this acknowledgement packet at T4 and can use these four time values to calculate clock drift and propagation delay. Node A can then correct its clock so that it synchronizes to B. In this sender-initiated approach the sender synchronizes its clock to that of the receiver.

The time synchronization process begins with the root node first sending a time-sync packet after which receiving nodes initiate the message exchange with the root node as described above in the synchronization phase. Once the nodes receive acknowledgement from the root node they adjust their clocks to that of the root node. This process is carried out until all nodes are synchronized with the root node.

Aggregation Queries

TinyDB supports aggregation of data which in turn allows reduction of data prior to transmission. The main difference between such queries in TinyDB and SQL is that the output to a query is a stream of values for the former while it is an aggregate value for TinyDB. Each aggregate record will have a <group id, aggregate value> pair and is time-stamped with an epoch number, all aggregates used to compute a complete aggregate record share the same epoch number. The aggregate operators allowed are the same as that allowed in a normal relational database system, for example AVG, MAX, MIN which compute the average, maximum value and minimum values respectively for the attribute they are applied over.

<u>Temporal Aggregates</u>

The TinyDB system acknowledges that aggregates over values within a common sample interval are not the only type of aggregate values that a user may want. In some cases a user will need to perform some type of temporal calculation. For example, users of a habitat monitoring system may want to monitor the temperature as a frost moves through a geographical area. This can be done by measuring the maximum temperature over a period of time and reporting that temperature at set intervals. In TinyDB this can be implemented as a *sliding window* query, for example:

```
SELECT WINAVG(TEMP, 60s, 5s)

FROM SENSORS

SAMPLE PERIOD 1s
```

The above query would report the average temperature over the last 60 seconds every 5 seconds with a sampling rate of once per second.

<u>Event-Based Queries</u>

TinyDB also supports event-based queries which are generated either by another query or by some part of the operating system, the language provides an EVENT clause for that purpose and the code that generates the event is compiled into the node.

The EVENT and STORAGE POINT commands were added to the original TinyDB SQL-like query language to allow more than just basic passive querying for simple environment data. It allows sensors to also detect and initiate automatic events in the case of the EVENT command and to store a streaming view of recent data in the case of the STORAGE POINT command [49]. An event in TinyDB is generated either explicitly using the EVENT clause or via another query or even a lower-level part of the operating system.

<u>Continuous Queries</u>

Continuous or lifetime-based queries are also supported using the LIFETIME clause as well as nested queries and offline delivery queries which allow data to be logged for non-real time delivery. Materialization points are used to implement offline logging.

<u>Query Optimization</u>

Queries in TinyDB are first parsed at the base station and then optimized to select the ordering of joins, selections and sampling. The optimizer determines the lowest power consumption which takes into account not only the cost of data acquisition but also processing and radio communication. Data acquisition, however, is the main source as it includes actions like sampling of sensors and transmission of query results. Query optimization, therefore, is focused on reducing the number and cost of data acquisition.

Once a query has been optimized it is issued into the network in a binary format, where it is instantiated and executed.

<u>Query Dissemination and Routing</u>

TinyDB uses semantic routing trees (SRT) to help determine whether a query is forwarded or not. Query

1.
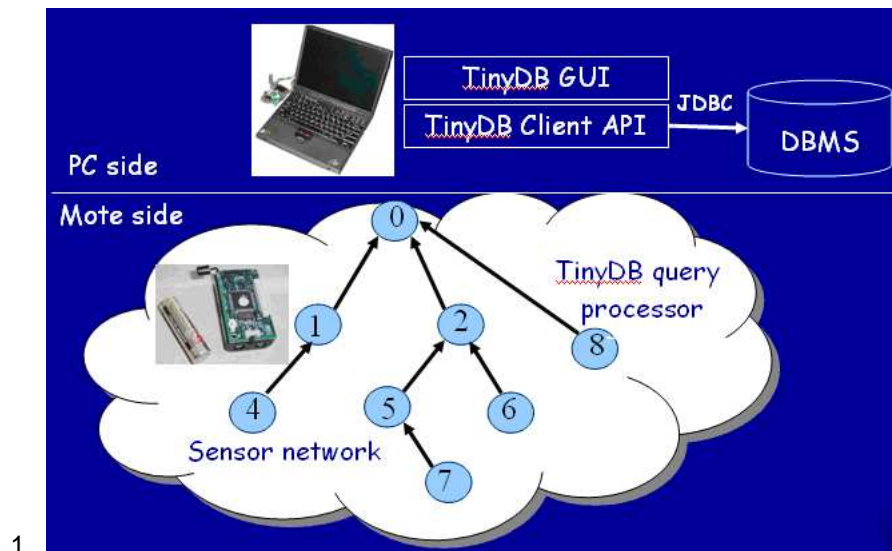
Figure 2.12: TinyDB architecture taken from [69]

dissemination and routing will be discussed later in more detail in Chapter 3.

Query Processing

After a query has been optimized and disseminated it is executed by the query processor. Execution follows a simple sequence of sleep, sampling, processing and delivery. Nodes sleep for as much time in an epoch as possible to conserve power and wake up to sample sensors and deliver results. The nodes are synchronized so parent nodes can ensure that child nodes are awake to process messages. Results are sampled and filtered based on the plan provided by the optimizer and then routed to the aggregation and join operators further up the query plan. In terms of data delivery TinyDB allows prioritizing through three schemes, *naïve*, *winavg* and *delta* for simple selection queries. The *naïve* scheme uses the first-in first-out (FIFO) queue processing technique to empty the queue, and drops tuples if they do not fit in it. The *winavg* scheme also uses FIFO but instead of dropping tuples, averages the two results at the top of the queue to make space. In the *delta* scheme, a tuple is assigned an initial score determined relative to the most recent in time and processes the highest score tuple first. If the queue is full the lowest score tuple is dropped. Figure 2.12 shows the TinyDB architecture.

In summary, the TinyDB approach does allow quite a good deal of flexibility in the types of queries that can be processed and shows increased efficiency in terms of power consumption using a number of query optimization techniques. Some further work needs to be done on optimization of multiple queries as these are not addressed as well as implementing more sophisticated data delivery prioritization techniques. The authors stress, however, that with TinyDB using the ACQP method it is essential that data delivery, query language and optimization techniques take into account the underlying hardware capability and semantics in order to increase the likelihood of successful deployments.

Add-ons aimed at simplifying the deployment and development of applications for wireless sensor net-

works, are also being developed. The Tiny Application Sensor Kit (TASK) is built on top of TinyDB for that purpose [17]. This kit contains a relational database used for storage of sensor readings, a server to act as a proxy for the network on the internet as well as a front-end that facilitates data selection and recording. Other architectures with a similar purpose also exist, for example jWebDust described in [21].

Although useful, like the other query-based approaches in use TinyDB has its drawbacks. The queries that can be issued are limited to those that target 'low level' sensed values on nodes in the network or those requiring the computation of simple aggregates like average, maximum, and minimum over some attribute. It does not allow the creation of nested queries or even nested, aggregate queries. In addition, the query language used cannot easily express spatio-temporal characteristics which are an important aspect of the data generated in WSNs. Therefore, there is still room for improvement and expansion in a query processing system that follows the same model but able to issue and process more powerful queries.

2.2.6   Active Query forwarding (ACQUIRE)

Common to all of the query-based approaches described above is the clear distinction between the dissemination phase when the query is sent out and the response stage when results are sent back to the querying node. ACQUIRE described in [108], does not distinguish between these two stages but rather issues what is called an *active query*.

An active query, in addition to the simple queries for an attribute or attributes, also includes nested queries. Each subquery can be a query of interest in a different variable or value. The active query is propagated through the network node to node. At any point in time, an active node (the node carrying the active query) uses data from a set of neighbouring nodes within a look-ahead of x number of hops and tries to resolve the query or part of the query if it can. The number of hops used can vary, but has been analyzed experimentally to determine what the most effective value is. The experiments in effect measured the average number of nodes from which new information is obtained during query forwarding based on different values for x and set that as the *effective look-ahead*.

Query Forwarding

Let us assume that a query Q with S sub-queries is issued by a node n. The look-ahead value is set to x which means that each sensor can request information from sensors within x hops from it. On receiving the query, the node does a number of things.

First, it checks whether its local state is up-to-date and if it is not will send a request to sensors within x hops; this request is then forwarded hop by hop. The sensors receiving the request will forward the information they have to the active node n. Second, using this information n resolves the part of the query that it can and selects a node to forward the query to. Selection of a node to forward to can be random or use other information that would most likely lead to further resolution of the query [108]. The idea is that as the query moves through the network it gets progressively smaller as it resolved piece by piece. Once the query is completely resolved the active query becomes a *completed response* and is routed back to the original querying node. The path it follows can either be the reverse path or the shortest path to the querying node. Figure 2.13 gives an illustration of the ACQUIRE mechanism at work.
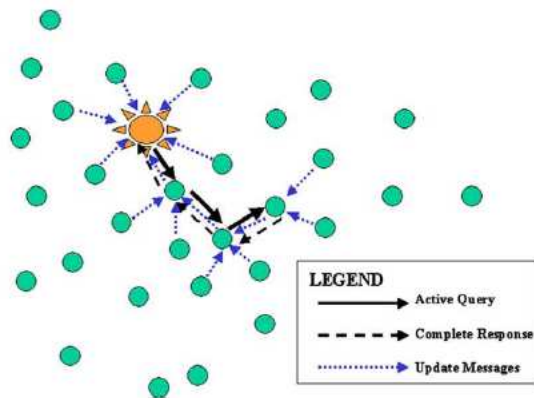
Figure 2.13: Illustration of the ACQUIRE mechanism given a one hop lookahead. At each stage of the active query propagation the node carrying the query using information gained to partially resolve the query, taken from [108].

Analysis of ACQUIRE has been restricted to networks exhibiting a regular grid topology. Given these restrictions, however, ACQUIRE worked better than flooding-based querying (FBQ) mechanisms like Directed Diffusion as well as had a 60-75% savings in energy consumption when compared to Expanding Ring Search (ERS) [108]. In ERS, a querier first requests information from all sensors *x* hops away where *x* is initially equal to one. If the query is not completely resolved at this stage it will send a request to *x*+1 hops away and so on. ACQUIRE is most suited to complex one-shot queries for replicated data and needs to be analyzed on more realistic networks where the topology is irregular or dynamic.

The reported evaluation of ACQUIRE is further limited as it uses mathematical modelling to analyse the performance of the mechansim in terms of energy costs. It does not address implementation. Further, the complex queries although including nested queries do not include queries where spatial or temporal characteristics may be of interest.

2.2.7   Agent Sensor Queries Language (ASQue)

[83]sets the basic framework for the Agent Sensor Queries Language (ASQue) which aims to allow the easy construction, execution and processing of queries within an ad-hoc sensor network. ASQue is specified using a form of first order logic and as it is meant to be the only application level interface to the hardware within the network a number of restrictions are imposed on the language. First, it has to allow a portable implementation suitable for multiple WSN architectures; second, it has to be energy-efficient in terms of its implementation; third, it has to eliminate the generation of error messages and unnecessary query responses. In effect, the language is designed in such a way that its algorithms are not unnecessarily resource intensive.

[83] describes the abstract syntax of ASQue which consists of only two sentences, a query sentence and a response sentence. The language is simple given the resource, power and communication bandwidth limitations of the network and its design specifically addresses the idea of eliminating runtime checking

and the desire to ease the implementation of static analyses.

A query in ASQue has the following structure:

```
query(q_recipients,

interested_parties,

query)
```

where `q_recipients` evaluates to the set of sensors which should receive the query, `interested_parties` are those sensors which should receive responses to the query and `query` is the query itself. To keep ASQue simple and self-consistent each of these components (and those of responses) are statements in first order logic (with equality).

A response in ASQue is a two-tuple:

```
response(r_recipients,

response)
```

where `r_recipients` are the nodes to which the response is addressed and `response` is the response itself. Again, both are statements in first order logic with equality.

Of critical importance with the ASQue querying mechanism is the need to issue 'correct' queries. This is not only in an effort to reduce resource wastage but needed because of the lack of runtime checking. With no runtime checking, however, comes the need for a user of the language to be confident that queries constructed are indeed correct and meaningful. It is critical that the accuracy of queries between parties be assured and that a 'trusted' relationship exists between the host issuing the query and the sensor nodes receiving it for processing. Ideally, sensor network developers should be able to formally prove certain properties of a query before issuing it in a deployed network. Such properties may include type safety, bounded resource use, an absence of deadlocks and livelocks, and so on.

In addressing these needs, the work on the ASQue mechanism focused on formal specification and validation of the language and a compiler was put forward as the first stage in achieving that goal. [29]details the specification and validation of a parser for the language through design of a parser for the language. The development of ASQue as an applicative language for query-based systems was seen as a viable approach for a number of reasons.

First, it promotes the need for the information extraction mechanism to display ease of use and to clearly express the user's requirements, a view supported by many of the query-based mechanisms in use today. A language based directly on a well-understood logic has the advantage of being simple to write and understand (for humans) and symbolically manipulate (for computers) [30].

Second, given the limitations of the network, including those of resource, communication bandwidth and power, the language is designed to eliminate the need for runtime checking as a way of maximizing efficiency in utility of sensor network resources. Eliminating runtime checking in turn motivates the need for sound analyses at compile time.

Third, with no runtime checking there is the need for a user of the language to be confident that queries constructed are indeed correct and meaningful. It is critical that the accuracy of queries between parties

be assured. Without this assurance, issuing queries could be risky in terms of potential resource wastage and using the language under those circumstances could not be justified. The language was designed with the idea that verification and validation would be done at compilation time in an effort to address these issues.

2.3 Macroprogramming Approaches

In the applicative query approaches described above, the user must have some knowledge of what type of queries or operations are to be issued as well as what raw data needs to be targeted in order to retrieve the information. Although information is retrieved, in many cases, the user still has to make sense of what it means and whether it is relevant or not. For example, a user could issue a query in TinyDB requesting the average temperature over the network for a particular period of time. Once the results are received, however, the user would then have to look at this 'information' in deciding whether it indicates that a fire has started. If it does, the user then has to decide what to do next, for instance, issue additional queries. The ability to create a 'program' where events are anticipated and response (further queries) issued is not allowed in the system.

Macroprogramming approaches to information extraction aim to treat the wireless sensor network as a whole by directly specifying global behaviour instead of programming individual nodes. It attempts to address the issues raised above in the form of a global program that is defined in a high level language that can capture operations going on in the network at a global level. A number of systems have been developed which incorporate global abstractions and in some cases couple them with node-level abstractions in creating useful macroprograms. In this section some of the more popular macroprogramming systems will be examined.

2.3.1 Node-level Abstractions

The underlying principle of macroprogramming is the creation of a global program that can capture network level operations. These macroprograms ultimately have to target node level data in order to function globally. A number of node-level abstractions have been proposed in the literature for modelling node level data and are used by macroprograms to target the data within the network. The data is not accessed on a low level, node by node basis but through this abstractions that provide access via logical 'collections' of nodes. The logical node-level abstractions proposed are usually based on one or more attributes within the WSN. This can be a dynamic attribute like a sensed value (temperature for example) or a static attribute like geographic location (in the case of stationary nodes). Following is a description of some of the more popular node-level abstractions put forward in the literature.

Hood

[127]describe a neighbourhood abstraction called a *hood* (which is defined by a set of criteria for selecting neighbours) and a set of variables to be shared within the hood. Hoods are mainly defined geographically with nodes being included in hoods based on the number of hops from a node. For example, a hood could be a one-hop or two-hop neighbourhood, over which temperature readings are shared or a one-hop
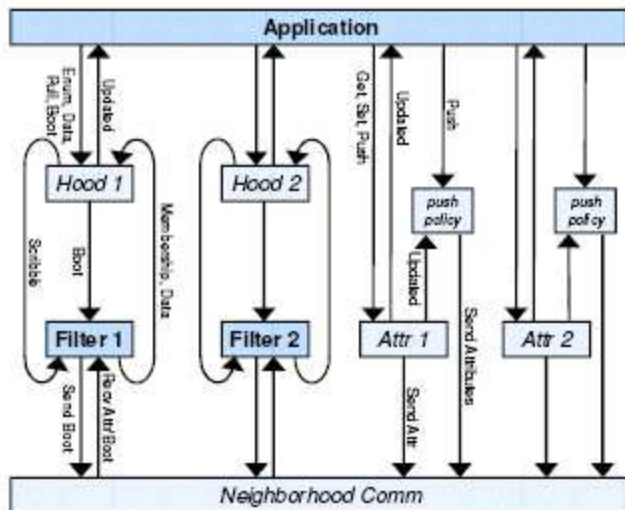
Figure 2.14: Component model for an application with two neighbourhoods and two shared attributes taken from [127].

neighbourhood over light and temperature readings. A node can therefore define multiple neighbourhoods over different variables.

Whitehouse uses a broadcast/filter mechanism to allow data sharing and neighbourhood discovery. Shared attributes are broadcast and receiving nodes cache any attributes of interest from valuable neighbours. A neighbour's value is based on how the neighbourhood has been defined. For example, a node may define a routing neighbourhood that caches location information of valuable routing nodes [127]. Once a neighbourhood has been defined and attributes broadcast, interested observers add the broadcasting node to its neighbour list and cache the attribute or attributes of interest. Figure 2.14 shows a model for an application with two neighbourhoods and two shared attributes.

Abstract Regions and Region Streams

[125]propose *abstract regions*, spatial operators used to hold local communications in regions which can be defined in terms of radio connectivity or geographic location. A region in that sense would serve as an identifier for neighbouring nodes who can then share data, identify each other, and reduce data among them. Nodes could then be manipulated via their common interface. An abstract region could therefore be defined as a relationship between a node and its neighbours and the main aim of this work is to move away from the need to construct low-level mechanisms for routing and data collection and instead focus on a higher level interface that is flexible enough to allow a user to implement queries. Such a system would not, for instance, have to consider routing, data dissemination or state management in order to function effectively, however it would still allow a programmer to create applications that can adjust to changing network conditions as well as adjust energy consumption to improve accuracy and latency.
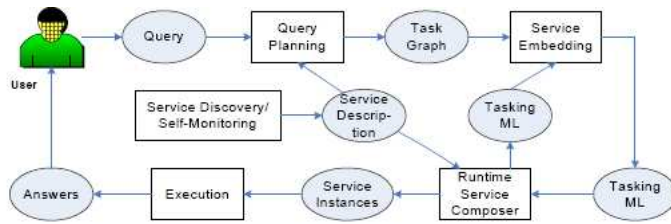
Figure 2.15: The semantic streams query processing model taken from [128].

2.3.2 Semantic Streams

Semantic streams, a system proposed by [128]allows a user to issue queries over *semantic* values **directly** without identifying what data should be targeted or what operations should be used on that data. It allows the user to interact with the sensor network using declarative statements, for example, "I want the ratio of cars to trucks in the parking garage". There is no need to construct code to actually check the data for the existence of cars or trucks rather components referred to as inference units are used to facilitate interpretation of sensor data. This is in direct contrast to other approaches that issue queries over raw sensor data like TinyDB and Cougar, and is of interest as it is similar in principle to the work proposed on complex queries in this research.

The Programming Model

The semantic streams framework uses the semantic services programming model which contains two main elements, inference units and event streams. An event stream is simply a flow of asynchronous events, each of which represents some real-world entity (for example, in the car park query given earlier, a car detection has properties such as the location at which it was detected, its speed and direction). An inference unit is a *process* that operates on an event stream. It infers semantic information about its environment and either adds it as a property to an existing event stream or generates a new one. Figure 2.15 shows the semantic streams model within its service-based architecture.

As a stream moves from a sensor through inference units its events acquire new semantic properties. A more detailed example, a vehicle detector service, follows.

The Query Language

Each inference unit is specified using a first-order logic description of the semantic information it needs in its input stream and that it adds to its output streams as well as any relationships between the two. A markup language (the Semantic Streams markup and query language) is used to construct a logical description of that semantic information. A number of predicates are used to describe sensors and services.

sensor(<sensor type>, <region>) - the sensor predicate defines the type and location of each sensor.

service(<service type>, <needs>, <creates>)

needs(<stream1>, <stream2>, ...)

`creates(` <stream1>, <stream2>, ...) - the service, needs and creates predicates describe a service, the semantic information that it needs and creates. In query processing these are treated as rules and their pre- and post-conditions.

`stream(` <identifier>)

`isa(` <identifier>, <event type>)

`property(` <identifier>, <property>) - the stream, isa and property predicates describe an event stream and the type and property of its events.

An example of a sensor declaration is as follows:

`sensor(` *magnetometer*, [[60,0,0], [70,10,10]])

This defines a magnetometer that covers a three-dimensional cube defined by the pair of 3-D coordinates.

An example of a service would be:

`service(` *magVehicleDetectionService*,

`needs(` `sensor(`*magnetometer*, *R*)),

`creates(` stream(*X*), isa(*X, vehicle*),

`property(`*X, T, time*), property(*X, R, region*)))

This vehicle detector service uses a magnetometer to detect vehicles and generates an event stream with the time and location in which the vehicles were detected.

Once a set of sensors and services have been declared the user can begin to issue queries. An example of a simple query would be:

`stream`(X), `isa`(X, vehicle)

This query would result in true if a set of services could be composed to generate events X that are known to be vehicles. All known possible service compositions would be generated. To constrain the result more predicates could be added. For instance,

`stream`(*X*), `isa`(*X, car*)

`property`(*X*, [[10,0,0], [30,20,20]], *region*)

This would restrict the result to cars found in the region described by the pair of 3-D coordinates given.

Query Processing

An inference engine is used to determine which sensors and services will provide the required semantic information. The inference engine employs the backward-chaining algorithm [93]using the pre-conditions and post-conditions of the services. An attempt is made to match each element of the query to the post-condition of the service; if this is successful the pre-conditions are added to the query. Once all pre-conditions have been matched to actual sensor declarations (without pre-conditions) the process terminates. In other words, the process terminates once all inference units' requirements have been met by physical sensors. The variant of backward-chaining used in service composition creates an instance of

each service and reuses any existing instances whenever possible. A description of a real-world application of the semantic streams framework is described fully in [128].

In the semantic model described, all services and their descriptions are maintained in a central repository or query server. Resource usage is optimized because the inference engine reuses services when possible, reuses resources and operations and the query language used allows great flexibility in how the query processor executes queries. The language also allows the user to specify constraints. When a user issues a query as an event stream the query planning engine generates a task graph which is then assigned to a set of nodes in the network. The service runtime on each node then accepts the graphs and instantiates services as required, resolves any conflicts between tasks and available resources and then executes the query.

One major limitation of the system is the semantic markup language used. Although suitable for simple examples it cannot capture more complex applications. Another drawback of the system is that the query processor cannot reason at runtime and so two applications which may possibly have to access the same device, for instance, would not be able to run simultaneously. It does not address sensor network issues that are not semantic transformations. For example, routing data between nodes in a sensor network would not change the semantics of the data being routed. Semantic Streams, can not differentiate between different routing algorithms and so cannot take advantage of their features if needed.

2.3.3 The Regiment Macroprogramming System

Regiment, proposed by [90]is a functional programming language for sensor networks. Its data model is based on the concept of region streams, described in Section 2.3.1, where sensor nodes are represented as streams of data that can be grouped into regions for the purpose of in-network aggregation or event detection. Collections of nodes, therefore, can be represented both spatially and temporally. A region stream is used to describe a collection of nodes with a logical, topological or geographical relationship. For example, a programmer may be interested in nodes which fall within certain 3D coordinates. The corresponding region stream would represent all sensor values for those nodes.

Operations allowed on region streams include *fold* which aggregates values across nodes in the region to an anchor node and *map* which applies a function over all values in a region stream. A *fold* requires inter-node communication while a map does not.

An example of a simple Regiment program is as follows:

let aboveThresh (p,x) = p>threshold

read node= (read_sensor PROXIMITY node, get_location node)

in centroid (afilter aboveThresh (amap read world))

Regiment's syntax is similar to Haskell [10]where functional applications are written as f x y. Regiment, as is the case with other functional programming languages, allows functions to take functions as arguments.

In the above example amap is a function that takes as an argument, the function read and applies it to every value in the region stream world. afilter filters out any values that do not match those

represented by the `aboveThresh` function.

The example program creates a region stream over the `proximity` sensor value of all nodes in the network. Each node's location is also included and any values below a threshold given by `aboveThresh` are omitted. The `centroid` function computes the spatial center of mass of the set of sensor readings so that we can determine the location of the device that generated the readings.

The Regiment compiler converts the high level program into node-level code that targets an abstract machine model called the token machine. This is an intermediate language that links token handlers to a token name which represents a task to be executed by a sensor node once it receives a token of that name. A token may be generated locally or through a radio message. The token machine language described by [88]elaborates on these concepts and defines the Token Machine Language which is based on an abstract machine model called Distributed Token Machines (DTMs). Typed messages with a small payload (a fixed size buffer) are referred to as token messages and used by DTMs for communication. Each token has an associated token handler which is executed when a token message is received by a node. DTMs, in addition, allow for concurrency, state management and communication. The aims of TML are to provide abstractions for key requirements in the sensor network including data dissemination and communication and provide a framework within which complex algorithms, such as those that allow in-network aggregation, can be implemented.

[89]describe the Regiment Macroprogramming System which extends the work done on the Regiment language and token machine interface. It uses 'signals' which represent attribute values for nodes, for example, the temperature read by a sensor, and groups signals into regions which are then used as a programming abstraction upon which different operations can be performed (aggregation for instance). Region membership is not static and may vary over time, due to changes in sensed values, node or communication failure or new nodes being added to the network. Key here is that the system abstracts away details of storage, communication and data acquisition from the user and allows the compiler to map global operations on regions and signals to local network structures [89].

The Regiment system has been evaluated in a number of simulated chemical plume detection macroprograms with a network size of 250 nodes over an area of 5000 x 5000m. The results demonstrated the flexibility of the system in maintaining good communication performance [89].

2.3.4   Kairos

In contrast to some programming approaches that focus on providing high level abstraction for local node behaviour in a distributed computation, the Kairos macroprogramming system [41] focuses on abstractions for specifying global behaviour. This distributed computation encompasses the entire network but uses a centralized approach. The abstraction considers the sensor network as a collection of nodes that can be tasked at the same time from one program. The programming model is based on shared-memory-based parallel programming models over message passing architectures and three programming abstractions are made available in the system. Figure 2.16 shows the architecture of the Kairos macroprogramming system.

Programming Abstractions
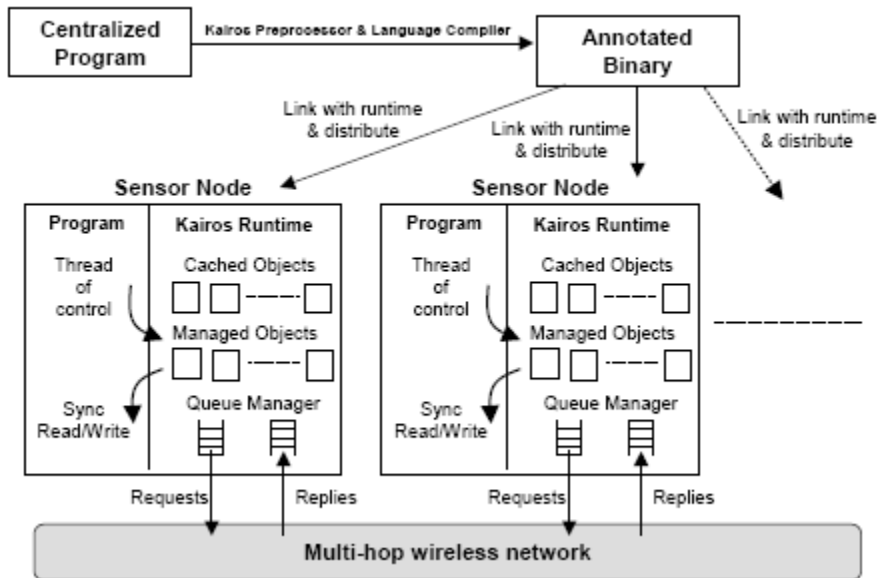
Figure 2.16: The Kairos programming architecture taken from [41].

Kairos provides three abstractions for manipulation by the programmer. The first is the *node* abstraction where programmers are allowed to manipulate individual nodes or lists of nodes. Nodes have integer-based identifiers and are represented by a node datatype which makes available operators like equality, ordering and type testing. There are also functions made available for manipulating lists of nodes.

A second abstraction is the set or list of all one-hop neighbours of a specified node which is made available using a get_neighbours() function. This abstraction is quite similar to the region and hood abstractions and it reflects the natural construct of radio neighbourhood of the nodes in the network.

The third abstraction is remote data access which allows a programmer to read from variables at a named node using a variable@node construct. In Kairos only a node may write to its variable although multiple nodes may have read access.

Programming Mechanism

A distinguishing characteristic of the Kairos system is that a programmer writes a single centralized program representing the distributed computation. First, the program is preprocessed to produce annotated source code which is then compiled into a binary. During this stage, the Kairos pre-processor identifies and translates remote data references to calls to the Kairos runtime. The binary can then be distributed to all nodes in the network using a code distribution facility. Although the initial program is a global level one, after compilation a node-level version is created that details what an individual node's functions are, when and what data remote and local it manipulates.

When a copy of the node-level program is instantiated and executed on a node, the runtime exports and manages variables that are owned by a particular node but are referenced by remote nodes.
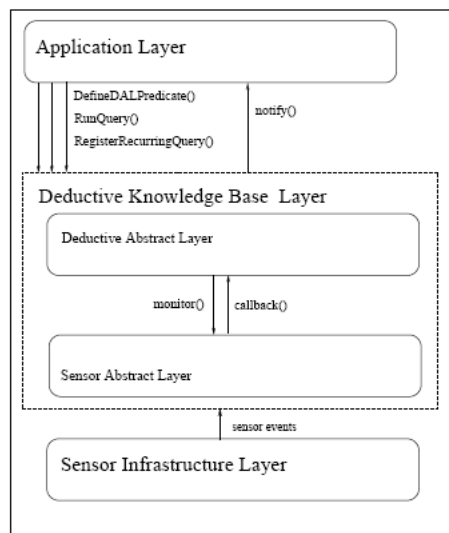
Figure 2.17: System architecture, taken from [62].

Kairos has been used to implement programs for a variety of problems from routing tree construction to vehicle tracking [41].

2.3.5   Knowledge-Representation for Sentient Computing

An interesting approach is proposed by [62]called a scalable knowledge representation and abstract reasoning system for Sentient Computing. Sentient Computing promotes the view that an application can be made more aware by perceiving the environment and reacting to changes in it. A gap exists, however, between the level of abstraction in the knowledge of the sentient world such a system requires in order to function and the low-level data produced by sensors in the environment. [62]propose the creation of a deductive component that would interact with the low level data, perform some type of reasoning function in order to deduce a higher-level abstract knowledge. This knowledge can then be used by the application layer. The deductive knowledge base layer therefore is really a domain specific language component positioned as a type of middleware layer between the application and hardware layers. A diagram of the system architecture is displayed in figure 2.17.

Knowledge Representation

An *event* in this context is defined as an occurrence of interest taking place instantaneously at a specific time. The *Sentient* environment refers to the physical environment. The *current logical state* of the Sentient environment includes all known facts about the sentient environment from an *initial event* to a *terminal event*. These events can include any events of interest to the *Sentient Application layer*.

The *Sensor Abstract Layer* (SAL) keeps a low-level view on the current logical state as sensors in the environment update it through sensed events.
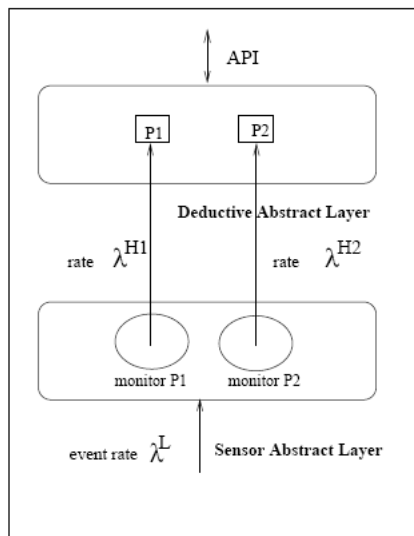
Figure 2.18: Interaction between DAL and SAL, taken from [62].

The *Deductive Abstract Layer* (DAL) maintains a high-level or abstract knowledge of the current logical state through its interaction with the SAL.

The two layers use a *monitor-callback* communication scheme to interact. The application layer issues a monitor call which causes the SAL to filter through to the DAL those low-level changes that affect the abstract knowledge in the DAL. The DAL, therefore, does not have to monitor all the data and is updated at a much lower rate than the SAL. This is depicted in figure 2.18.

Both DAL and SAL were described using first order logic and queries which are quite similar to SQL SE-LECT statements are used by the application layer to retrieve the stored knowledge about the *Sentient* environment. Experiments conducted with a prototype implementation showed that the two-layered architecture where the low-level and high-level interactions are separated (SAL and DAL) was more efficient than a single-layered one for a number of reasons [62]. First, queries executed in the DAL were of a simpler form with fewer conditions so used less computational resources. Second, the knowledge update rate triggered by *assert* or *retract* commands in the DAL is lower than that of the SAL making DAL more computationally efficient. Experiments with more complex queries have not been carried out.

2.4   DISCUSSION

The three approaches to information extraction surveyed show systems and techiniques with varying degrees of ease of use, flexibility, informational complexity and expressiveness. While the querying approaches are easier for a user to manipulate they are restricted by the lack of expressiveness of the query languages they provide for requesting information. This in turn restricts the type and level of information that can be retrieved. Although it is clear that simple queries for attributes and simple aggregates are possible and extensively demonstrated in the deployments to date, informational query-based systems that

go beyond that (for example spatially and temporally aware queries) are practically non-existent. Query-based systems are also limited as they do not allow users the flexibility to target smaller areas of interest within the network but in effect force the user to task the entire network in order to get a response, an issue in terms of energy efficiency. An advantage however is that these systems have been extensively used, the languages used familiar to users of sensor network systems and are relatively easier to design and implement when compared to other information extraction mechanisms.

In contrast, the agent-based approaches are more expressive and flexible in terms of what functions can be performed and the ability they incorporate to make decisions while in the network. Agents can act autonomously, multiple agents can run on a node at the same time, and multiple applications can co-exist in the network (this can be a problem with many query-based and macroprogramming systems). Mobile agents can move, clone themselves and can act to deal with unexpected changes in the environment. Agent-based systems, however, are hindered by the difficulty they present for non-expert users to design and program. It is also apparent that despite the many agent-based techniques and models proposed in the literature, actual deployments have been minimal and restricted to very small networks (less than 10 nodes) with agents at most just pieces of mobile code with minimal autonomy or decision-making capability.

The Macroprogramming approach appears to be one solution to the problem of the limitations of query-based systems while at the same time promoting the expressiveness inherent in agent-based systems. Some researchers consider macroprogramming an extension to query-based applicative approaches and attempt to adhere to the ease of usability mantra of query-based systems, while at the same time providing users the ability to access higher level information. The reality, however, is that the systems are not nearly as intuitive as many of the SQL-based querying systems and require a learning curve for the programmer. In addition, the approach seeks to eliminate the need for low-level programming on the node but in reality has to provide node-specific abstractions which undermine rapid program development. Finally, the high abstraction level although advantageous on the one hand in terms of shielding the user from the underlying workings of the network (radio communication and network topology, for example) presents a challenge when constructing compilers that need to cater not just for computation but node level communication as well. Macroprogramming therefore presents similar challenges to the agent-based approaches.

Overall, the approaches to information extraction examined highlight that trade-offs must be made between ease of use and expressiveness coupled with flexibility (for instance in terms of decision-making capabilities within the network) in selecting a suitable information extraction system. Clearly, the query-based and macroprogramming approaches are useful in a number of applications but raise a number of challenges as outlined above. There is a definite gap between the ease of use provided by the more popular query-based mechanisms and the expressiveness of the global and node level abstractions provided by macroprograms as well as the flexibility of agent-based systems. For purposes of this research, and indeed for any applied research in WSNs, the idea of usability and utility are paramount. To that end, the development of a declarative query-based system is put forward as being the most viable option. Given the limitations already described above a number of features have to be incorporated. In order to make the querying approach more attractive, the development of a query language that allows a user to more richly express high level information requests that can target the full breadth of collectable information presented by sensors in the network is promoted. Complex queries are proposed as suitable constructs

that can allow the expression of spatio-temporal characteristics and requests requiring more involved in-network interactions to generate information. A second objective is to incorporate within this query processing system useful elements that will aid the processing of these complex queries. Promoted is the use of abstractions already employed in some macroprogramming approaches to facilitate resolution of this higher level information requests. This *hybrid* approach, it is anticipated, will retain the simplicity and ease of use of the traditional query-based approaches while simultaneously allowing the inclusion of useful logical abstractions provided by macroprogramming approaches to facilitate dissemination, processing and resolution of more powerful queries than those available in query processing systems today.

Chapter 3

Architectures,Algorithms and Processing Techniques

Within an application or WSN deployment, the information extraction approach used is influenced to a great extent by the type of architecture the WSN exhibits. In the literature, particularly prior to the widespread use of formal WSN design methodologies, the word architecture was used as a wrapper for a number of interests at different levels of the WSN protocol stack (for example, application level designs, hardware configuration, routing and MAC layer protocols.) These designs were then put forward as all-inclusive WSN architectures, whether query processing or agent-based, without separating out the concerns at various layers. Practitioners designing a system usually implemented network lower level protocols that would allow for their particular application requirements to be met. There was therefore a tight coupling of application-specific components to network and lower level protocols.

In this chapter, a distinction is made between a number of concerns that impact the design of a WSN processing system and these are described in the next three sections.

The first part of Section 3.1 reviews some common WSN topologies found in existing or proposed WSN systems and details their evolution as application needs have changed. In many systems the efficiency of the information extraction method used is influenced to a large extent by the routing method implemented. In many cases routing is incorporated into the information extraction mechanism itself or vice versa. The second part of Section 3.1 investigates some of the routing protocols that drive many of the WSN applications in use. Particular attention is paid to those that have been used in query processing systems in the literature and how they impact the generation of information within the WSN. With this in mind, at the end of the section, a suitable group of protocols for the system under development in this research is identified and a network model for the research work is put forward.

Section 3.2 investigates a number of candidate query processing architectures, identifies their key features and makes the case for the query processing architecture proposed for the complex query processing system under development in this research.

Finally, Section 3.4 investigates some popular information processing techniques used in information extraction systems, particularly query-based systems. At the end of the section techniques applicable to

| Application Layer |
| Transport Layer |
| Network Layer |
| Data Link Layer |
| Physical Layer |
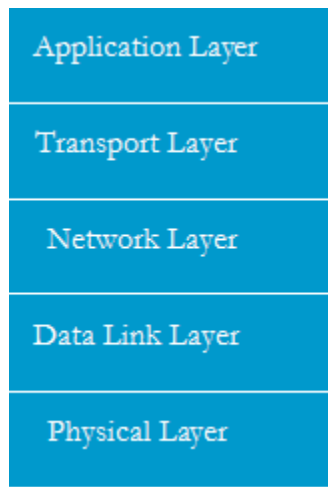
Figure 3.1: Typical protocol stack for a WSN.

the system under design within this research program are identified.

3.1 WSN Topologies and Routing Protocols

Figure 3.1 shows a diagram of the typical protocol stack for a WSN. Energy and bandwidth constraints mean that all layers of the networking protocol stack have to be energy-aware. At the physical and data link layers this involves using energy-aware MAC protocols, low duty cycling and power-aware radio communication. At the network layer, the main concerns are energy-efficient data routing, that is, setting up routes in the network and relaying data from sensor nodes to a gateway [2]. Energy efficiency is important in maximizing the lifetime of the network.

A number of topologies sometimes incorporating underlying routing protocols have therefore been put forward specifically to address these issues. Topologies and protocols have further evolved as a result of changing application needs. Some key topologies are described in the next section.

3.1.1 WSN Topologies

The most basic WSN topology is the centralised, sink-based topology sometimes referred to as a *flat* or single-tier architecture where nodes in the network are homogeneous, that is, identical in terms of hardware complexity and battery power [18, 60], and last but not least, bandwidth management.

Data collected by the nodes are directed toward the sink or base station (usually the only 'node' more powerful both computationally and in terms of energy capabilities) using single or multi-hop communication [56]. Figure 3.2 shows a diagram of a typical flat WSN topology. This configuration brings a number of challenges particularly if there are a large number of nodes. These include management of energy consumption, energy optimization, routing, information gathering and general management of the sen-
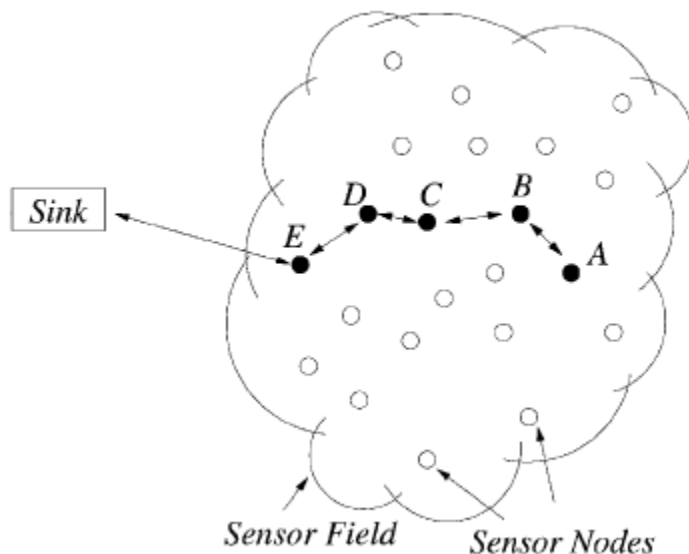
Figure 3.2: A typical flat WSN architecture, taken from [3]

sor nodes themselves [18, 60]. The University of California at Berkeley's Redwood forest deployment [38]is one example of a WSN system exhibiting a flat architecture. The 33 node deployment used TASK [17], a self-contained sensor network system based on the TinyDB query processor [71]to monitor the microclimate (temperature, relative humidity and solar radiation) of a redwood tree over a 44 day period.

Whilst centralised, sink-based topologies are relatively simple to support and implement, they do not fulfil a topical requirement of WSN designs: scalability. Scalability has been described as one of the key design requirements both for conventional communication networks as well as wireless sensor networks. As the number of nodes increases with a flat topology, however, the sink node may become overloaded leading to increased latency as well as severe energy usage [2]. As a result of the apparent problems posed by flat networks, hierarchical heterogeneous architectures were proposed. The simplest example of this type of network consists of two layers: the first contains groups of homogeneous nodes, called clusters, connected to a dedicated micro-server or cluster head [56]. Cluster heads are sparsely distributed and serve as aggregators of data and managers of nodes within their individual clusters. They also serve as communicators both to a gateway or sink node and with other cluster heads in accomplishing application goals. Figure 3.3 shows an example of a hierarchical architecture.

The introduction of cluster heads with special functionality (usually supported by enhanced hardware) gives the hierarchical structure some advantages over the flat topology particularly with respect to energy consumption.

Note: In this respect, multihop communication has been identified as a more favourable strategy over single hop, given that energy consumption is directly proportional to the square of the distance [57]. Hence, there is potentially more energy conserved with multiple short hops from node to sink, for example, as opposed to a direct long hop from a node to the sink [22]. A problem occurs, however, with the nodes
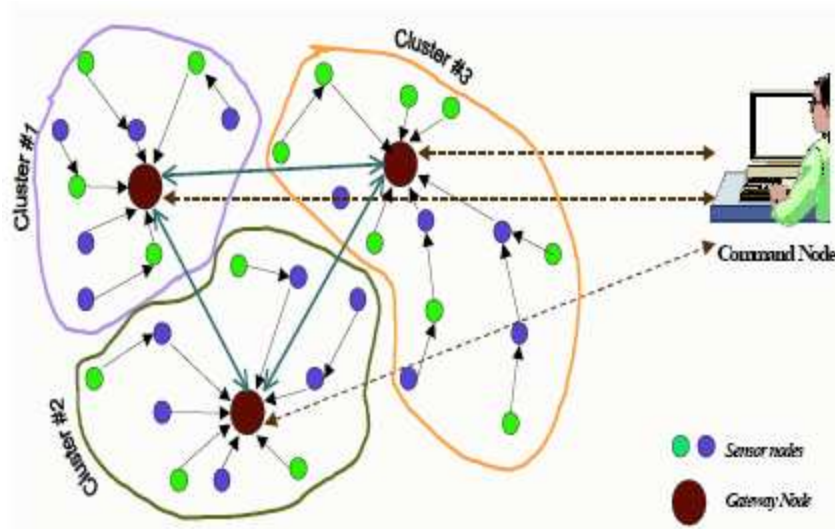
Figure 3.3: A hierarchical, heterogeneous WSN architecture, taken from [134]

closest to the sink since they are under heavier traffic and more likely to have their energy drained more quickly. High energy consumption can also be a problem if clusters where multihop communication is used are large.

Cluster heads provide additional functions like caching and forwarding of data to the required destination as well as performing data aggregation and fusion in order to decrease the number of transmissions to the sink or gateway. This is of particular importance in data gathering networks. It has been shown that for some applications, using a hierarchical structure can bring about significant improvement in performance in terms of reliability, longevity and flexibility of the network [56]. The Great Duck Island habitat monitoring application [73]is one example of a deployment that used a hierarchical architecture.

Building on the basic two-tier hierarchical model, varying multi-tier architectures have been put forward to capitalize on the apparent advantages. An example proposed by [43]consists of three layers where nodes in a *sensor layer* sense the environment, pass that data to a *forwarding layer* whose purpose it is to facilitate communication between other forwarding nodes and nodes within the highest tier, the *access layer*. Nodes in the access layer communicate with the network backbone in transferring information.

Another three-tier architecture is the "data mule" architecture [112]where agents are used to periodically collect sensor data and deliver those to wired access points when within range. Finally the SENMA architecture [78]proposes a novel two-tier architecture where the upper layer consists of mobile access points that are used for data acquisition from homogeneous sensors within the sensor layer. Sensors communicate with mobile agents who periodically move within radio communication range eliminating the need for multihop communication. This architecture, because of the reduction in multihop communication, showed a significant improvement in energy efficiency.

3.1.2 Routing Protocols for WSNs

Given the unique characteristics of WSNs the underlying routing protocol used is an important consideration when looking at information extraction. In many cases the information extraction mechanism and how efficiently it works depends to a great extent on the routing method used. In some cases the routing mechanism itself includes techniques for query dissemination and data aggregation as in Cougar [133]and ACQUIRE [108]. Routing is therefore an important aspect of information extraction in WSNs.

Many routing protocols are currently being used in a variety of wired and wireless networks [103]. Although protocols can be classified in different ways, for example, based on the network structure or perhaps on the protocol operation [4]many follow the address-centric (AC) model where routes are found and followed between pairs of addressable nodes. In mobile ad-hoc networks, AC protocols are used widely (MANETS) [63]. Here each source independently sends data along the shortest path to a sink. This path is usually based on the initial route a query took to get to that source node. Although MANETs are similar to WSNs in that they both involve multi-hop communication, their routing requirements are quite different for a number of reasons.

First, communication in a WSN comes from multiple data sources to the sink instead of just between a pair of nodes. Second, redundancy in sensed data is common in WSN since data is being collected by multiple sensors which may be sensing the same phenomena, a fact which data-centric protocols take into account in devising optimal routes. This is not a major consideration in MANETs. Finally, in WSN the major constraint is energy making it essential that data communication rates be made as efficient as possible, much more so than in MANETs.

Given these reasons among many others, the traditional end-to-end protocols used for MANETS are not appropriate for WSNs. Some alternative protocols propose a different model, the data-centric model, where sources send data to a sink, but the content of the data is examined and some processing (whether aggregation or fusion) is executed on that data en-route to the sink. The result in using such protocols is that a better transmission/information ratio is achieved. In the next section some data-centric routing protocols will be examined in more detail.

The TinyDB Model for Query Dissemination and Routing

In TinyDB, a query is first broadcast to all nodes in the network from the root or base station. A node receiving the query will then decide if the query applies to it and/or if that query needs to be broadcast to its child nodes in the routing tree. A semantic routing tree (SRT) is maintained which contains information on the child nodes and will help in determining whether a query needs to be forwarded down the routing tree or not. If a node is aware that its children cannot answer a request for data, it does not forward the query.

A SRT is constructed in two phases. First, a SRT build request is flooded through the network until all motes have received it. The request will include the attribute name, A, over which the tree should be built. As the request moves through the network individual nodes may have more than one possible choice for a parent node. If a node does not have children it chooses a parent node p from its collection of possible parents and forwards a *parent selection message* to p. If a node has children it forwards the request

to them and waits for a reply. It records the value of A for each child and when it has heard from all its children selects a parent, sending a *parent selection message* as described before. This message, however, will include the range of values for A which the node and its children cover. The parent node will record this information and choose a parent in a similar way. This process continues until the root node is reached.

As an example, we consider a query *q* with a predicate over an attribute *A* arriving at a node *n*. Considering that the routing tree has already been established, *n* checks to see whether any of its children values for *A* are in *q*'s range of values for *A*. If that is the case, the node prepares to receive results and forwards *q*. If there is no overlap between those *A* value ranges *n* does not send *q* on. Irrespective of the state of its children, *n* will also determine whether the value of *A* applies locally and if it does will start executing . If it does not *q* is ignored.

According to [71], SRT allows more efficient query dissemination and collection of query results in cases of constant attributes. In cases where attributes are highly correlated in particular areas of the routing tree, SRT can reduce the number of nodes required to disseminate queries and forward results.

Directed Diffusion

[54]propose and describe directed diffusion (DD), a data-centric protocol for requesting and retrieving data in the network. In directed diffusion data is named [47]using attribute-value pairs and a request is sent into the network as an *interest* for named data. As a result, gradients are created within the network and serve to draw data matching those interests. These are referred to as events and they start to move towards the instantiators of interests along different paths.

Directed Diffusion takes into account a number of challenges. First, the need for scaling to larger networks with not just hundreds but thousands of sensors and second, the awareness that sensors can fail, lose power or may be unable to communicate due to other factors. The mechanism, therefore, has to be robust enough to withstand those challenges particularly reducing energy consumption within the network.

Naming

Tasks or requests in DD are represented as lists of attribute-value pairs. For example, an animal tracking task could be described as follows where rect represents a subregion within which the request is to be analyzed:

```
type=four-legged animal
```

```
interval=20 ms
```

```
duration= 10 seconds
```

```
rect=[-100,100,200,400]
```

This describes an interest for data matching the attributes indicated. If a sensor detected an animal matching the attributes it could generate data that looked like the following:

```
type=four-legged animal
```

```
instance=elephant

location= [125,100]

intensity=0.6

confidence=0.85

timestamp=01:20:40
```

Each attribute in this naming scheme has a range of values, for example type as used above represents mobile objects in the area of interest (vehicles, animals, etc.). The value of the type attribute in the interest described above would be any corresponding to a four-legged animal.

Interests and Gradients

A named task, like the one described in the example above, is called an *interest*. An interest is injected into the network from a base station or sink node. The interval value will determine the event data rate. In the above example the data rate would be 50 events per second and the task is recorded by the sink. After the time specified in the duration value has elapsed, the sink will no longer retain a record of that task's state.

For each active task, a sink node will periodically broadcast an interest message to all of its neighbours. Initially, the interval attribute will be quite large and the data rate for response low as the message is more of an exploratory broadcast aimed at determining whether any nodes can respond.

Each node maintains an interest cache and each item in that cache is distinct. Two interests are distinct if the type or interval differ or if the rect attributes are disjoint. An entry in the cache has a timestamp indicating the time the last matching interest was received. Each entry also has gradient fields each with a data rate field requested by the specific neighbour initiating the interest. The data rate field is derived from the interval attribute of the interest.

When a node receives an interest it checks the cache to determine whether a matching entry exists. If not, it creates an entry in the cache, instantiates its attributes based on the information in the received interest and then sets up a gradient towards the neighbour from which the interest was received. This gradient will have the specified event data rate found in the received interest. When a gradient expires it is removed from the interest only when all gradients for a particular interest have expired will the interest itself be removed from the cache. A recipient node may resend the interest to some of its neighbours but may suppress the interest if it recently sent a matching interest. In the simplest case the interest is re-broadcast to the node's neighbours. In this manner interests are diffused throughout the network.

Data Propagation

In addition to processing interests as described above a node will also instruct its sensors to start collecting data. When a data message is received from neighbours the node checks the interest cache to determine whether a matching entry exists. If it is not found that data message is dropped. If a match is found, the node will check the data cache associated with the matching interest entry. If a match for that data message is found in the data cache, the data message is dropped. If no match is found the message is added to the data cache and the data message resent to the neighbouring nodes.

<u>Reinforcement</u>

When an interest is first injected into the network it specifies a low event-rate notification. If matches are detected low-rate events (one per second) are sent towards the sink. Once the sink starts receiving these events it **reinforces** one in an effort to draw higher data rate events. This is usually done by using some type of data-driven rule, for example a rule may be to reinforce any neighbour from which a previously unseen event is received.

Reinforcement is achieved by the sink resending the original interest but this time with a higher data rate. A neighbouring node receiving this interest will observe that it already has a gradient to this neighbour and that the data rate specified is higher than that in its cache. This node will then reinforce at least one of its neighbours by checking its data cache, identifying, for instance the first node from which it received the last event matching the interest.

The aim of reinforcement, therefore, is to promote optimal data paths for data propagation using interest transmissions to nodes considered worthy of reinforcement.

Directed diffusion is especially suited to continuous queries and shows better energy efficiency than omniscient multicast and flooding schemes [54]. When average dissipated energy was calculated as a function of network, for some sensor networks using DD average dissipated energy was 60% of that observed with omniscient multicast. The effectiveness of the data aggregation and caching techniques for improving efficiency are examined further in [63]. The authors have identified that future work would look at examining its efficiency under other scenarios such as congestion.

Some work has been done in both using directed diffusion and adapting it to specific application and application scenarios.

[39]designed and implemented a sensor database system using TinyDB [71]as the front end of a system interacting with a network running directed diffusion. As is the case with directed diffusion, queries were injected into the network as interests and replies sent back as data messages. No networking code was necessary as the system relied entirely on the protocol for routing queries and replies.

[136]proposed the Geographic and Energy Aware Routing algorithm (GEAR) which uses energy-conscious neighbour selection techniques to route packets to a target area. This protocol, they suggested, could be used in conjunction with DD to make it more energy-efficient. The idea is that by eliminating the initial interest flooding that occurs and routing interests to the target region instead, significant energy savings could be made.

Finally, [46] proposed a new version of DD that seeks to minimize the effect of flooding in large networks. With DD interest messages are disseminated throughout the network setting up gradients. Their approach, however, eliminates the exploratory phase and sends data immediately along a preferred gradient. This gradient is determined to be the first neighbour from whom a matching interest was received. In this version of DD no reinforcement occurs, rather, the lowest latency path is *implicitly* emphasized.

Rumor Routing

In contrast to directed diffusion in which initial flooding of a query toward an event is required, [16]propose the rumor routing algorithm aimed at providing an alternative between query flooding and event flooding protocols. It aims to lower energy consumption by minimizing the number of hops required for transmitting information in long-range transmissions. Data is retrieved based on the event observed rather than on any available geographic information.

Each node in the network maintains a list of its neighbours and an events table containing information on all the events that a node is aware of. An event is described as an abstraction, encompassing anything from a set of sensor readings to the processing capabilities of a node. When an event is witnessed the node adds it to its events table and generates an agent.

Agents are used to create paths leading to an event once it occurs. An agent is described as a long-lived packet with travels through the network distributing information about local events to distant nodes. Each agent carries a list of all events it has encountered as well as the number of hops to the event. When the agent arrives at a node it informs that node of all events that it has encountered. Conversely, if the node has information on any events that an agent does not have the agent's list is updated as well. There is therefore synchronization of the two lists. The agent also contains an event table and synchronizes it with every node it visits. Since query target events, a query, once it arrives at the node, will be routed along these agent-generated paths if they match the event of interest. Queries target events and so any node may generate a query. If a node has a route to the target event it will forward the query along that route. If a route is not known the query will be forwarded to a random neighbour where it will again check if a target even is in that node's event list. If the event is not found the query is again forwarded on randomly. This process continues until the node that has observed the desired event is reached or the query expires. If the node originating the query determines that it has not reached a destination, it can retransmit it, terminate transmission or flood the query as a last resort.

Simulations show that the rumor routing algorithm is more energy-efficient than flooding-based algorithms in delivering queries to events in large networks particularly where geographic information is not available. Where a *time to live* (TTL) of 1000 was used with a small number of agents, 31, the rumor routing algorithm successfully delivered more than 98% of all queries with an average cost of 92 cumulative hops per query, approximately 1/40 that of a network flood. In addition, the algorithm can be tuned to support varying delivery rates, route repair and query/event ratios and handles node failures effectively. However, it is shown to work well only where the number of events is small but still has to handle the overhead needed to maintain agents and event tables.

Information-Driven Sensor Querying (IDSQ) and Constrained Anisotropic Diffusion Routing (CADR)

[26]propose the IDSQ mechanism for dynamic querying of sensors and CADR for routing of data in WSNs, in an effort to reduce bandwidth consumption. IDSQ allows selection of sensors able to provide the best information among all available sensors while CADR routes the query to an optimal destination. This is an instance of where the querying and routing mechanisms are closely linked.

Other approaches to querying and data routing like directed diffusion [54]use the distance values between

nodes in determining a cost of communication in terms of number of RF hops, to determine what routing path should be taken. IDSQ, however, uses both communication and information costs to decide on a path. CADR then routes the query to the optimal destination following an information gradient, optimal destinations are computed using a composite object function in conjunction with a number of algorithms. This method can be more energy-efficient and energy-aware than directed diffusion [26].

Other Approaches

[8]propose new algorithms that focus on query dissemination and information retrieval on a hierarchical network architecture. Although similar to the distributed query processing environment of TinyDB, there are some differences. First, the architecture supports XML and is therefore open to web-based monitoring applications. Queries are constructed using the XQuery language [20]and routing trees can be built concurrently and support either the same or different queries. Another difference is that queries are maintained in data packets and trees are constructed parallel to dissemination of the query.

[59]developed an analytical model of an energy-constrained, data-centric information routing protocol that uses game theory to model intelligent sensors. They called their approach sensor-centric as they take the view that the sensors in a routing game would act to maximize their own payoffs. Nodes acting like intelligent agents would have to cooperate to determine optimal routing paths. Payoffs here are described as benefits to the network brought about by the sensor's action less individual costs. In modelling data-centric routing they not only use data-centric techniques like aggregation and attribute-based naming but also consider the *importance* of the data itself in selection of data transfer paths. In their view popular data items which represent successful query matches should be routed preferably via more reliable paths as opposed to less important data. They do this by attaching a number value to the data retrieved from a sensor and use this value to reason about what path that data should take.

3.1.3   Discussion

Although protocol development is beyond the scope of this research, it is important to have an awareness of what protocols are used within the query processing systems in use and their impact on the information extraction mechanism. Data routing, for example, is an important consideration when looking at information extraction as it affects the overall efficiency of the mechanism. For purposes of this research, a suitable cluster-based routing protocol that facilitates the implementation and testing of the logical abstractions (the main focus of the research) will be incorporated into any experimental network setup.

The choice of architecture for a WSN system appears to depend strongly on the application requirements. Pure data collection applications tend to work with hierarchical topologies with cluster heads aggregating and relaying data as it is generated. In such systems information processing requirements are simply sense data collection or at most calculation of aggregates. This is not always the case, however, as some WSN systems like the redwood forest deployment [38]which for the most part exhibit a flat topology are also used for data collection. In such systems energy conservation is either not a major concern or techniques for optimising query processing are incorporated to minimize energy consumption.
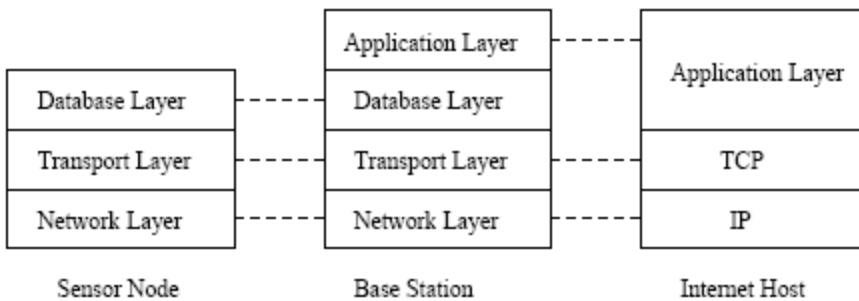
Figure 3.4: A database-based query processing architecture taken from [32].

For the work in this project a cluster-based toplogy (single or multihop) will be the primary focus as such topologies appear to be more amenable to the logical region-based query processing approach proposed for investigation. Region leaders can be considered somewhat similar to clusterheads in terms of functionality therefore enabling a mapping of the logical construct to the network construct. It is expected that regions, in this type of architecture will be easier to implement and test experimentally. The network model proposed for this research has a hierarchical topology, randomly deployed sensors and will use a cluster-based routing protocol that allows dynamic cluster formation. The cluster-based protocol selected will also support both inter-node communication and communication with the external world. The network model is described in greater detail in Chapter 4.

3.2 Query Processing Architectures

3.2.1 Database-Type Architectures

Database-type query processing systems are some of the most popular put forward in the literature. [32]propose a generic database-based query processing architecture for sensor networks (see figure 3.4) and describe the components they feel are required at each layer to make the architecture practical .

Application Layer: at the base station this layer provides an interface for posing of queries to the sensor network. These queries can be represented either as an SQL-type message or even a SOAP-like web service. The application layer in the base station is designed to communicate with the TCP/IP stack and so query results can be easily accessed by an internet-based host.

Database Layer: on the base station, the database layer serves the application layer by receiving queries from it and returning results to it. The database layer on the base station contains a number of components including:

- Parser - creates the query tree based on the query received from the application layer.

- Catalog Manager - maintains the relational schema, location and distribution of sensors and monitors nodes' status, for instance, node power levels and node connectivity.
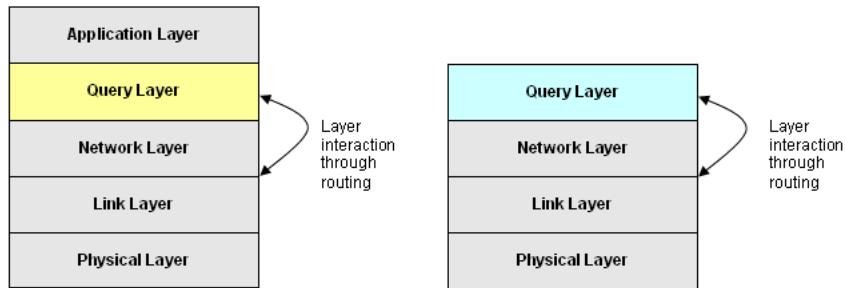
Figure 3.5: A high level view of the query processing architecture used in Cougar andTinyDB.

- Query Optimizer - generates the optimized query execution plan.

Due to resource limitations, the database layer's function at node level is much more limited. It receives the query tree either from the base station or another node and may need to further optimize the tree based on local catalog information. The database layer returns results in the form of relations[1].

Transport Layer: this layer provides for efficient end to end communication. At the base station it should be able to communicate with TCP entities on internet hosts although alternative methods may be needed for nodes since nodes are address-less.

Network Layer: this layer should provide energy-efficient and data-centric routing algorithms. The routing decisions made should be based not only on the current network conditions but also the given query execution plan.

Example Systems

Query processing systems like TinyDB [71]and Cougar[133] are just some of the database-type systems put forward in the literature. They all follow to varying extents the architecture put forward by [32]. The query processing architecture is similar for both, consisting of server-side software running at a base station and responsible for parsing and delivering queries into the network, as well as collecting the results as a stream out of the network.

The architecture of the TinyDB system, both server-side and node-side, is illustrated in figure 3.5. Architecturally the query layer sits between the application and network layers in the case of the server side component (base station) and sits on top of the network layer at node level. Like in figure 3.4 the application layer allows the construction and posing of queries with the database layer, here referred to as the query layer, below parsing the query, constructing query execution plans and delivering the results to the network. The query layer at the server side also collects the results for presentation to the application layer.

The query layer houses a number of components some of which have already been described in the architecture put forward by [32]and are illustrated in figure 3.6.

---

[1]A relation represents information as tables consisting of tuples or rows made up of attributes or columns. In the context of a sensor network, a relation can be a static table that is stored to disk or a virtual table where its state is continuously changing.
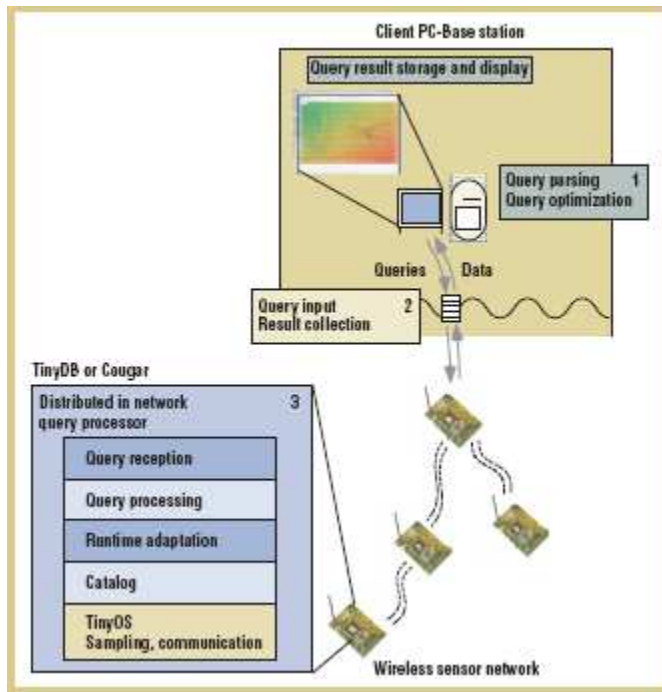
Figure 3.6: A detailed view of the query processing architecture of TinyDB and Cougar taken from [37]

The query layer is the backbone of the query processing system and provides a number of critical functions:

1. The first is the processing of the query itself. The query layer uses sophisticated techniques like catalogue management, query optimization to abstract the user from the physical details of the network in processing queries.

2. The second is to perform in-network processing to reduce communication given the need to preserve resources like energy and bandwidth. To do this the query layer generates different query plans with different tradeoffs for requirements such as accuracy, energy consumption and latency.

3. With both TinyDB and Cougar, the query layer interacts with the routing layer to facilitate in-network processing. With traditional routing, the network layer on a node will automatically forward packets to the next hop towards the destination with the upper layers, completely unaware that data packets are moving through the node. In implementing in-network aggregation, for example, the query layer needs to be able to communicate with the network layer when it wants to intercept packets destined for the sink or leader node. In the processing system described above, filters are used to first access the packets then modify and delete if necessary before passing on to the next hop onto the destination. The routing mechanism used in TinyDB is described in detail in Section 3.1.2.1.

Although they can probably be implemented on both hierarchical and flat network topologies, these database type approaches have so far been implemented on flat networks.

3.2.2 Middleware Architectures

Another approach to query processing in WSNs is that of query-based middleware systems. Two of the more popular are DSWare and SINA described in Section 2.2.3 and 2.2.2 respectively. Although also inspired by database systems in that they use SQL-based languages for construction of queries, these systems are considered middleware approaches as they also provide services that go beyond database query processing systems. Architecturally, these middleware approaches are similar to the database approaches described in section 3.2.1. In both cases, the middleware layer sits between the application and network layers and in addition to providing facilities for query processing, supplies the services necessary for ensuring adequate WSN functionality. A diagram of the DSWare architecture is displayed in figure 2.9 in Section 2.2.3 and the SINA middleware architecture is depicted in figure 2.8 in Section 2.2.2.

In the case of DSWare the application layer allows the construction of SQL-based event type queries which are then registered, parsed and execution plans generated. The underlying DSWare middleware layer is then responsible for registering and executing these events. The DSWare layer also interacts with the routing layer for improving network functioning, for example, improving power awareness.

SINA also comprises a middleware layer that sits above the network layer and provides an application programming interface to the middleware layer via SQTL scripts. The middleware layer runs on all nodes in the network and like the database approach it selects the most suitable distribution of the query based on the current network status and the type of query being executed. In addition to query processing functions, SINA also provides services for accessing sensor hardware and communication and supporting changing network topology (for example, sink mobility).

Both DSWare and SINA promote processing of aggregation-based queries and are suited to cluster-based topologies.

3.2.3 The Proposed Architecture for a Distributed Complex Query Processor

The architecture described in section 3.2.1 forms the basis for the complex query processing architecture put forward in this research. The two part architecture consists of server-side software which is accessed by the user and used for constructing and posing declarative-type queries and node-side software which is in effect the distributed complex query processor (DCQP). A key component of the architecture will be the Region and Query Management Layer which will be responsible for not only query processing but will have additional tasks related to the creation, maintenance and update of regions within the network.

A number of factors influenced this selection. First, a key feature of the distributed complex query processor (DCQP) proposed is ease-of-use and familiarity for users. An SQL-based, database-like approach is therefore preferred. Although attractive, a middleware approach was not selected as the aim with the DCQP is to create a system that is separated as far as possible from network level configuration issues. The idea is that the system will sit on top of a functioning network. Second, the need for in-network processing for resolution of complex queries dictates that a query layer that allows in-network processing of queries is essential. The approaches above have shown that distributing the query layer to all nodes and providing functions for query processing and optimization are effective. The DCQP query layer there-

fore, will allow the formulation of query execution plans, dynamic optimization of these plans based on the attributes of interest and node state as maintained by the regions. Third, the DCQP architecture will incorporate a query layer that is decoupled as far as possible from the underlying layers. The idea here is to make the system as portable as possible by not strictly linking it to any particular routing protocol, for example, but instead identifying features that would make particular protocols more suitable than others. Routing protocol selection to support the region abstraction was discussed in Section 3.1. The DCQP architecture is described in greater detail in Chapter 4.

3.3 In-Network Processing Techniques for WSNs

Experiments have shown that the major part of power consumption costs is due to communication rather than computation [48, 70, 94]. [72]describe **in-network processing** as the pushing of operations, particularly selections and aggregation of data, into the network to reduce communication. [133]describe it as the moving of computation from outside to inside the network in an effort to reduce energy consumption and improve network lifetime. The main goal in effect is to reduce communication in exchange for some form of computation within the network. In-network processing techniques are therefore accepted as being essential to improving energy efficiency in gathering information within the WSN and critical to any system that has improved network lifetime and energy efficiency as goals. The two most common in-network processing techniques used for reducing communication in WSN systems are packet merging and partial aggregation [37]. In addition, a number of other widely used techniques exist. These are discussed below.

3.3.1 Filtering

Filters are constructs used within the network (on nodes) to assist in processing [47]. A filter registers what kind of data it handles through matching and is triggered each time that type of data enters the node. Once it has been activated, the filter can manipulate the data by, for example, determining whether it is forwarded on or even generating a new message. Filtering is sometimes used in conjunction with data aggregation. [47]proposed using a filter to detect concurrent detections of four-legged animals from different sensors. It could then record what the desired interval was and ensure that only one response per interval, suppressing responses from other sensors. [130]take advantage of event properties of monitoring queries, and carry out data filtering during in-network processing.

3.3.2 Packet Merging

Packet merging, described by [37]combines a number of smaller records into a few larger ones in a bid to reduce the number of packets needing to be transmitted and in turn reducing the cost of communication as the cost of transmitting several smaller packets is greater than transmitting just one large packet. The Cougar query processing system described in section 2.2.1 is an example of a system that uses packet merging for reducing communication in processing some types of aggregate queries.

3.3.3 Partial Aggregation

Partial aggregation refers to the computation of intermediate results as readings are received by the node. Communication is reduced as this combined partial result is transmitted on instead of all individual records. Aggregation can be achieved in various ways based on the type of correlation that exists in the WSN. This may be spatial correlation due to physical proximity of nodes and therefore similarity in the readings, temporal if there is little variation in the sensed attribute based on the sampling frequency and finally, correlation in the data itself due to overlap in sensor coverage [102].

Tree-based Aggregation

In tree-based aggregation one node is designated the root node. A broadcast message is sent out with data on the ID of the node sending the message and its depth in the tree. In the case of the root the depth would be zero. When nodes receive this message they assign themselves a level by adding one to the value in the message received and assign the ID in the message as their parent. Broadcasting of the message continues until all nodes within range have received it and assigned themselves a level and parent. TinyDB (section 2.2.5) is one query processing system which uses a tree-based aggregation service called TinyAGgregation and this is described in more detail below.

Tiny AGgregation (TAG)

TAG is an aggregation service designed for TinyOS motes and is described in detail in [68, 37]. It is designed specifically for issuing and processing aggregate queries.

First, an aggregate query is constructed using a SQL-like declarative language as in the TinyDB system. For example:

```
SELECT AVG(volume), room

FROM sensors

WHERE floor=6

GROUP BY room

HAVING AVG(volume)>threshold

EPOCH DURATION 30s
```

The `EPOCH DURATION` clause here specifies the amount of time (in seconds) nodes wait before acquiring and transmitting samples. Queries use a table called `sensors` which is an append-only relational table.

The query is then injected into the network from a base station. Operators in the network implement the query and use the underlying routing protocol to distribute it. In returning a response, data is routed back via a routing tree that is rooted at the base station. As it flows from node to node readings are combined or aggregated according to an aggregation function and irrelevant data discarded.

There are a number of advantages to the TAG approach. First, a reduction in the number of communications needed to calculate aggregates when compared to aggregation that occurs at one centralized

location. Second, as data moves up the tree back to the base station nodes usually are required to transmit a maximum of one message. This is in direct contrast with centralized aggregation methods where the number of transmissions increases dramatically as data moves towards the root node. This can of course lead to a decrease in the lifetime of the network as battery power is quickly drained. Third, it allows aggregation even in networks where connectivity is intermittent or disconnections occur since disconnected nodes can reconnect by listening to other nodes' partial state records as they flow up the tree. This is possible since the partial state record includes information on the query that was issued.

TAG, therefore, presents a simple interface, flexible naming and generic operators for constructing aggregate queries and is not application specific as with other approaches using aggregation like Directed Diffusion [54]and Greedy Aggregation [55]. Further, it separates the logic of aggregation from routing details so that the focus is squarely on the application and leaves routing decisions to the system. This is in contrast to Directed Diffusion which puts aggregation mechanisms within the routing layer. The TAG approach allows a stream of aggregate values that change as sensor readings and the underlying network layout change and does this in an energy-efficient and bandwidth-efficient way.

TAG, however, does have limitations as it does not allow joins and cannot respond to events that occur within the network. The authors have indicated that future work will focus on developing an efficient way of aggregating the results of those event-based queries across nodes before transmitting that information to interested nodes.

Cluster-based Aggregation

In some cases nodes are in close physical proximity to each other and queries may need to retrieve information based on this spatial correlation. For example, a query may want to determine the average humidity over a particular area. Typically, clusters are formed consisting of nodes in close proximity based on some metric, for example, signal strength. Clusterheads are elected and act as the data aggregator and router for cluster members. SINA and Cougar, already discussed in Section 2.2.2, are examples of query processing systems that incorporate cluster-based aggregation in processing aggregate type queries.

3.3.4 Discussion

Given the importance of considering in-network processing when developing efficient WSN applications, this section examined some of the more popular techniques used. Data aggregation is most widely used, with the type of aggregation incorporated dependent on the network topology and to some extent the type of queries being issued. Systems that incorporate in-network processing and are therefore information-generating systems (as opposed to simple data gathering) have tended to be built on cluster-based topologies. In heterogeneous configurations, cluster heads can be configured to be more computationally powerful and are conducive to supporting data aggregation. There have, however, been information-generation systems based on flat topologies which allow tree-based aggregation. Where scalability is an issue, however, and where in-network processing is computationally intensive, reported systems have had the tendency to lean towards a cluster-based configuration, this being more energy

efficient and practical in creating a system with an extended lifetime.

Clustered networks have the advantage of conveniently allowing aggregation at the clusterhead and are most effective in networks that are static and where the cluster structure stays unchanged for a considerable period. With dynamic clusters, however, problems can occur with energy expended in continuously updating nodes in order to keep the clusters consistent with the underlying network topology. Tree-based aggregation is simple and useful but can lead to problems. Given a node failure in the network, for example, a packet lost at a given level of the tree can lead to all data being aggregated up that node's sub-tree being lost as well. This can lead to incomplete data making its way up the tree particularly if nodes only have one route to the sink and connectivity gaps occur.

With regards to this research, a cluster-based strategy will be used initially; however, further work may require that more efficient tree-based routing protocols be incorporated within regions (clusters), if necessary, to improve query processing within individual regions.

Chapter 4

Monitoring Applications

Military applications were the motivation for much of the initial research into WSNs with the Defense Advanced Research Projects Agency (DARPA) providing funding for a variety of projects from early 1990s to the present day. Today, however, the use of WSNs has expanded to encompass a large number of application domains. A particular group of WSN applications, denoted as *monitoring applications*, make up a large proportion of the WSN systems being deployed today. Though most monitoring systems are generally characterized by the need for attributes of interest to be observed, sensed and then relayed to a user, most WSN applications of this type show very little genericity in design. Rather, the WSN systems deployed are application-specific, with designs intimately linked to the particular application requirements. The next sections will identify some of the common types of monitoring application and describe examples of each, highlighting the information gathering model used in each example.

4.1   Habitat and Environmental Monitoring

Habitat and environmental monitoring form a particular set of monitoring applications that have grown over recent years. This is in part because of the great benefit they claim to provide to science and education as well as the fact that funding for improving science education is a priority. There are a number of WSN deployments in this category.

On the environmental side, the ARGO project [6]uses a sensor network to monitor the salinity and temperature of the upper ocean. The aim of the project is to generate a quantitative model of the changing state of the upper ocean while monitoring ocean climate patterns over the short to longer term. The data gathered is expected to act as input to ocean and ocean-atmosphere forecast models for data assimilation and model testing. Nodes are dropped from aircraft and cycle between the surface and a depth of about 2000m every ten days collecting data. When nodes rise to the surface, data collected by nodes are individually transmitted to a satellite.

The Great Duck Island project [73]is an example of a habitat monitoring application where a WSN was deployed to monitor the breeding behaviour of small birds called petrels. Scientists are interested in the

usage patterns of nests as well as the environmental changes in and outside of nests during the breeding season. Measurements are taken of humidity, temperature, pressure and light level. Nodes are clustered with each cluster connected to a base station via a long-range antenna. Sampling is done every minute and readings are sent directly to the base station which connects to a database via a satellite link.

Another environmental monitoring application is the GLACSWEB project [75]which uses a sensor network to monitor sub-glacier environments in Briksdalsbreen, Norway. Holes are drilled at different depths in the ice and sensor nodes carrying pressure, temperature and tilt sensors are deployed in them. The nodes communicate with a base station on top of the glacier which measures supra-glacial displacements. The collected data is transmitted via GSM with no information processing occurring within the network itself.

For all of the applications above, raw sensed values are sent via base stations or servers to a *user* and processing occurs at that point.

4.2   Agricultural Monitoring

Increasingly, WSNs are being used to monitor conditions that affect plant growth. These networks are used in precision agriculture which aims to improve crop yields, reduce pollution and monitor the general health of crops. Variables monitored include temperature, soil moisture, humidity and light and are usually referred to as micro-climate variables. The Climate Genie system [86]is one such commercial example and is used to monitor vineyards. Nodes are spread over the vineyard in a wireless mesh configuration and are equipped with sensors for measuring temperature, moisture and light. The data gathered is summarized (aggregates that reflect grape quality and vitality) within the network and sent via Wi-Fi, cellular or satellite to servers for viewing anywhere using a web browser. This system therefore incorporates some level of in-network processing although limited to the calculation of aggregates which are used in decision making by the observer.

Another system was developed to monitor in realtime field conditions including leaf moisture, soil temperature, soil moisture and CO2 [51]. Field monitoring servers (FMSs) similar to web servers were deployed in rice paddy fields, collected data automatically and transmitted the data for permanent storage in publicly accessible databases. Real-time data was made accessible via a web browser. Like the habitat and environmental monitoring applications described in section 4.1, raw sensed values are sent to the databases for storage and analysis occurs from that point on.

4.3   Structural Health Monitoring

Structural health monitoring (SHM) refers not only to the state of health of a given structure whether a building or bridge, for example, but also the detection of changes that may affect the structure's health in the future. There are two types of SHM systems:

- Systems that monitor disaster response after some catastrophe has occurred, for instance, an earthquake and

- Systems for continuous health monitoring which may check for signs of stress, monitor vibrations, wind, etc.

Disaster response systems are still a young area of WSN research. [131]describe a centralized WSN for structural-response data acquisition called Wisden. Wisden collects structural response data from a multi-hop network and relays and stores it in a base station. [66]also proposed a wireless monitoring system aimed at detecting damage to civil structures after a disaster (such as an earthquake) has occurred. In other research, technology developed by the CodeBlue project [27]is being used in the AID-N project [1]at Johns Hopkins Applied Physics Laboratory in developing systems aimed at disaster response.

For continuous, structural health monitoring applications the focus is on high sample accuracy with minimal distortion, high frequency sampling, time synchronization of readings and efficient data collection as opposed to energy efficiency through reduced power consumption. [118]describes a SHM system deployed on the Golden Gate Bridge. 64 nodes were deployed over a 4200ft long length and measurements taken of ambient structural vibrations. All collected data were relayed to a base station for analysis.

4.4   A Motivating Scenario

A WSN application frequently put forward is that of forest fire detection and monitoring [116].This application is attractive because forestry is a major industry in many parts of the world, and forest fires are a major cause of loss of wood (in the USA the annual average loss to fire is 17,000 km^2). Early warning of a fire event and the manner in which it spreads are hence necessary.

Currently, MEMS chemical sensors are available which can detect the volatiles produced in the early stages of a fire. Other useful information to firefighters would be air temperature and wind direction, for which sensors are easily available. These sensors may be readily interfaced to an existing wireless sensor node, indicating that such a system is easily feasible, at least in terms of the hardware required.

In theory, however, any practical forest fire detection system is likely to exceed the scale of present WSNs by a considerable margin (the largest WSN deployment to date, ExScal [7]fielded just over 1000 nodes). Without detailed knowledge of the application domain it is difficult to make sensible 'guesstimates' of system requirements. Even with a conservative estimate of 10m from the source of the fire being adequate for detection of preconflagration gases, this still equates to coverage of 100 nodes per hectare. For even a small forest (for example, Wyre Forest, Worcestershire, UK which covers 2634 hectares) this means in excess of 200,000 nodes.

Consider now a proposed deployment in which the sensor nodes are equipped with two types of transducer, an 'electronic nose' to detect the pre-conflagration gases and infra-red pyro-electronic sensors, which can detect the head of a fire.

In this situation, the initial detection of a possible fire will most likely come in the form of an event from the electronic nose. It is likely that this will be at some distance from the real seat of the fire, the gases being borne on the prevailing wind from the fire to the sensor. The next task is, therefore, to localise the fire. To do this the infra-red sensors are used to construct a heat map of an area surrounding the node which generated the original alarm.

Given the scale of the network envisaged in this theoretical scenario, it is difficult to see how such a search could be achieved efficiently using a conventional centralised query mechanism. For each detection cycle, 200 000 readings must be returned to the sink and processed. Following detection of the nascent fire, new queries must be generated and directed to the nodes in the area to be mapped. Finally, those nodes need to return the infrared data required for the map. A more efficient proposition would be for the initial event to be detected within the network, with the subsequent queries for the map data being generated locally, without returning data to the host.

A number of deployments have tackled some of the very issues described above but for practical reasons, the deployments have taken very different approaches to those proposed in the scenario described above. The FireWxNet system [45]a wireless sensor system aimed at monitoring weather conditions in wildland fire environments is one such example. The system monitored a variety of weather conditions that influence fire behaviour with an aim to using them to predict fire behaviour. The application, as is the case with most monitoring applications, was driven by a list of requirements acquired through consultation with both fire fighters and fire researchers. The implemented WSN system was deployed in the Bitterroot National Forest in Idaho (USA) and consisted of 3 sensor networks totaling 13 nodes, 5 wireless access points, 2 web cameras and 5 long range links.

The deployment was distinguished by the rugged environment within which the WSN had to function as well as the sparseness of the deployment itself. The authors note that sparse coverage was a deliberate choice aimed at strategically placing nodes to cover as much meaningful terrain with as few nodes as possible. Rather than scale, the focus was on creating an extremely robust design which included not just robust equipment but robust routing protocols as well. Once the system was launched, a number of challenges were encountered particularly given the harsh deployment environment but the system overall was a success. During its operation, over 80,000 data were streamed realtime, with operations applied post-collection to transform that data into usable information.

This real-life deployment presents a marked contrast to the motivating scenario and approach described above: the problems **are** well characterized given the input of domain experts; there is quite detailed knowledge of the application domain and what the corresponding WSN application requirements are. However, on-the-ground challenges usually mean compromises have to be made and as a consequence real-life monitoring systems so far, rarely aim to deploy such large quantities of nodes as put forward in the motivating scenario here.

The FireWxNet example highlights the limitation of the vast majority of monitoring applications today. They are limited in that they are conceptualized, designed and implemented as primarily data collection systems. In-network processing is absent and the systems can be considered automated to the extent that there is little user interaction. In essence, events are defined, queries may be deployed for sensed readings and all data is simply relayed to a centralized location for further analysis. Very few systems look beyond this simple sense-and-send model to incorporate some sort of analysis, in-network, prior to communicating results. Granted, with some systems it is difficult to define beforehand what events or processes may be of interest and these applications by their nature have to be more exploratory at least at the pilot deployment stage, with feedback influencing subsequent iterations. However, with some applications, as in the FireWxNet example, the problems are better characterized and therefore more amenable to some more sophisticated analysis or processing within the network. This could make the application not only more useful but more efficient as well.

Hence, the view put forward in this research is that in many of monitoring applications (the fire monitoring scenario being only one example) it is possible to define high level information requirements that could incorporate in-network processing techniques in resolving the requests. Besides the efficiency benefits in terms of reduced transmissions, this approach could transform a data collection system to an information generating system, extending the application scope while still fulfilling the basic application requirements.

As an example, an **informational** fire monitoring system would allow the processing of high-level queries aimed at tracking the fire. The queries would need to allow the expression of spatial and temporal characteristics and the querying system would need to support a level of autonomy in terms of some in-network decision-making by the nodes given the impracticality of streaming all data back to the sink. Such a system would perhaps allow a user to zoom in on particular problem spots and query for even more detailed information on-the-fly. Complex queries are proposed as extremely suitable for use in the scenario above as well as other monitoring application scenarios and a degree of autonomy is introduced through the implementation of in-network logical abstractions for query processing and resolution.

4.5   A Distributed Complex Query Processor

In achieving the goals of successful in-network processing and resolution of complex queries, a number of requirements for the proposed query processor have to be addressed. In Section 3.2 of Chapter 3, a number of candidate architectures were investigated as a starting point for a distributed query processing system that incorporates best practices for in-network processing and resolution of complex queries while at the same time allowing the incorporation of novel techniques and strategies for the types of queries proposed. The next subsections describe in more detail the design choices and assumptions made and give reasons for these selections.

4.5.1   The Network Model

Key to the research work is the creation of logical regions within the network. These regions constitute a critical component in making a decision on where a query should be disseminated, how the query should be processed within and between regions and where the complex query will be ultimately resolved. The regions will have one 'leader node' each which acts as a manager, data aggregator and communicator to a gateway as well as other region leaders when needed. This role is remarkably similar to that of clusterheads in network-level clusters [56]. Additionally, like some clusters, regions should, and are likely to be in the approach proposed here, dynamic. Consequently, it makes sense that these logical region leaders map to physical clusterheads. These requirements, therefore, directly influence the choice of network model for the proposed work.

First, the network model proposed has a hierarchical topology. Sensors are randomly deployed and the transmission range is identical for all devices. For simplicity, an ideal MAC layer is assumed and node death considerations are not taken into account at this stage. Also assumed is that a route has been established between nodes in the network and the gateway[1] node. To facilitate region-based routing, a

---

[1]Gateway node here is defined as the node from which a query is disseminated and through which results are acquired.
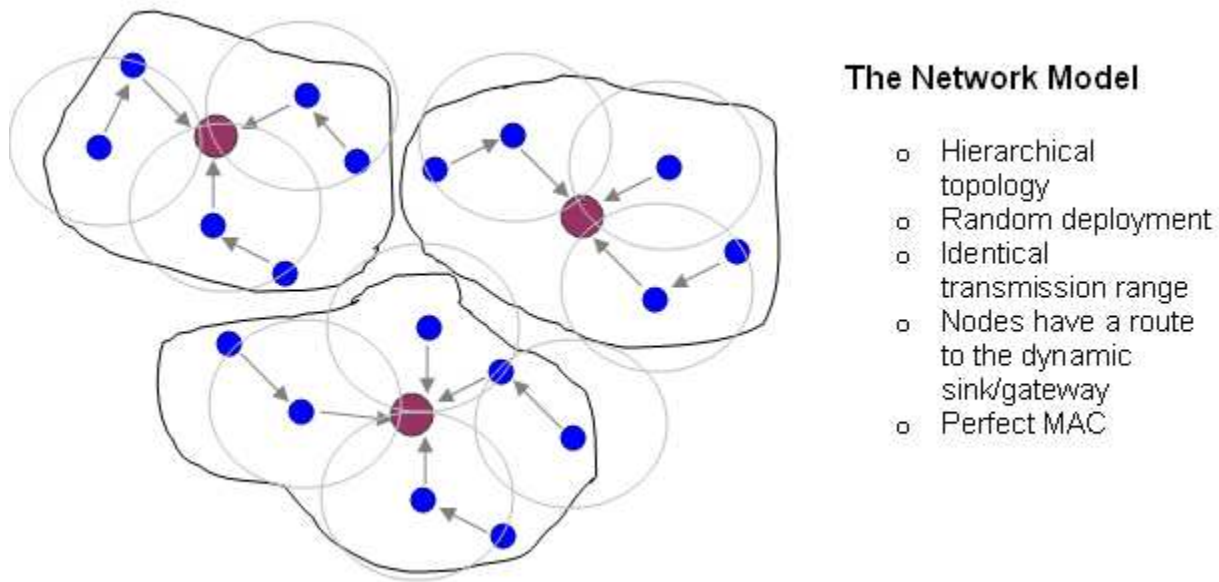
Figure 4.1: The proposed network model.

cluster-based routing protocol that allows dynamic cluster formation and supports inter-node communication and communication with the external world, is selected. Two possible choices identified so far are the Hybrid Energy-Efficient Distributed Clustering protocol (HEED) presented by [135]and which focuses on scalable data aggregation and increased network lifetime; and a dynamic clustering algorithm (DCRR) presented by [9]in which cluster heads are dynamically selected in the region where an event occurs according to their residual energy. Figure 3a gives a pictorial description of the network model.

4.5.2   A Distributed Complex Query Processing Architecture

Section 3.2.3 introduced the architecture proposed for the DCQP. It consists of server-side software which is accessed by the user and used for constructing, posing and parsing of declarative-type queries and node-side software which is in effect the distributed complex query processor (DCQP) implementing query processing functions in-network.

<u>Server-side Software</u>

This component will host an application layer which will allow the construction of queries with an underlying Region Query Management Layer which will be responsible for parsing of these queries and dissemination to the network. Additionally, the server-side software will allow the receipt of query results for both presentation purposes and for persistent storage. Figure 4.1 shows the server-side architecture.

A key output of the research work will be the creation of a system that will allow the user to either specify new logical regions as a component of a query being constructed or make use of regions already in

Gateways are not fixed; any node could potentially become a gateway at some point in time.
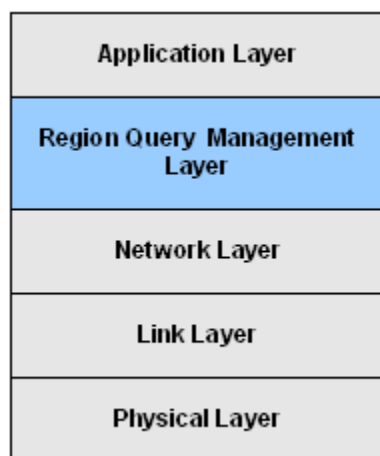
Figure 4.2: DCQP Server-side architecture.
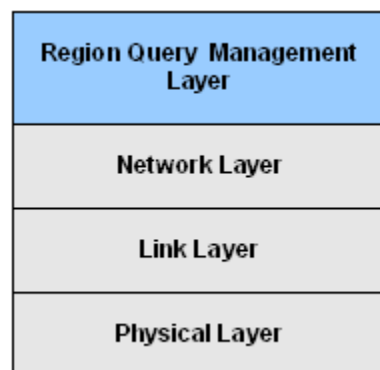
Figure 4.3: DCQP Node-side architecture.

existence within the network. This will involve the extension of a SQL-based query language to incorporate the region construct and the development of an associated parser for the language. The region abstraction will be a part of the query language used, and is essential to query dissemination as well as the creation of query execution plans.

Node-side Software

This component will also host a region query management layer which will from an architectural stand-point sit above the network layer (see figure 4.2). The node-side component is really a distributed query processor hosted by all nodes in the network.

The region query management layer will deal with query reception, dissemination, processing and delivery of results. In addition, this layer will need to manage the logical regions which are implemented as part of the complex query. This functionality will be implemented as a region management component of the query management layer itself.

Region management will include the creation, maintenance and updating of region information on each node within the network. This may involve, for example, a region leader upon receipt of a query, informing its members of a new attribute of interest; or perhaps the termination of a previously posed query along with its associated regions; or the handing-off of a region leader's duties to a backup leader when the node's energy level falls below a certain threshold. Once a query is received, execution plans are created dynamically based on the required region formation or existing regions which need to be accessed. Region management is therefore an integral part of query plan formulation and query optimization. The query plan is then executed. Once results have been acquired these are communicated to the server-side component.

As in the Cougar system described in Section 3.2.1, the query management layer may have to interact with the routing mechanism in the network layer in order to enable in-network processing which is a key feature of the region-based query processing approach proposed. This interaction, however, is anticipated to be minimal since a cluster-based routing protocol which maps the logical regions to physical clusters is proposed as a way of minimizing intrusion into the network layer by the query management layer. Figure 4.3 shows diagrammatically, the proposed complex query processing system architecture.

Initial simulations have already been conducted to assess the feasibility of the approach proposed in this work and are discussed in Chapter 5. Future work will identify a number of queries within the complex query category and processing of these using the proposed techniques will be analysed first in simulation. Based on the results, an integrated software/hardware implementation and deployment will be attempted. This will involve testing both the in-network processing techniques themselves as well as the construction of the queries using an identified and suitably modified query language.

4.5.3 Proposed Implementation Approach

The complex query processing system will be implemented on a suitable hardware platform which in the first instance will be the Gumstix [42]. The devices are currently accessible and the research will be aided by the creation of a gumstix-based testbed which will be up and running by the end of March 2008. The platform will support various communication methods including Bluetooth, Wifi and Zigbee, and will allow the implementation of a suitable routing protocol required to support the implementation of regions and processing of region-based queries.

The software will consist of a query processing system which will be housed by each node in the network. These will interact with sensor data acquisition services already handled by the underlying platform and data routing services. Each node will be responsible for maintaining data such as:

- a list of its one-hop neighbours.

- queries which it has received and have to respond to.

- attributes of interest specified in those queries of interest.

- regions it may be a part of and

- a path to its region leader(s).
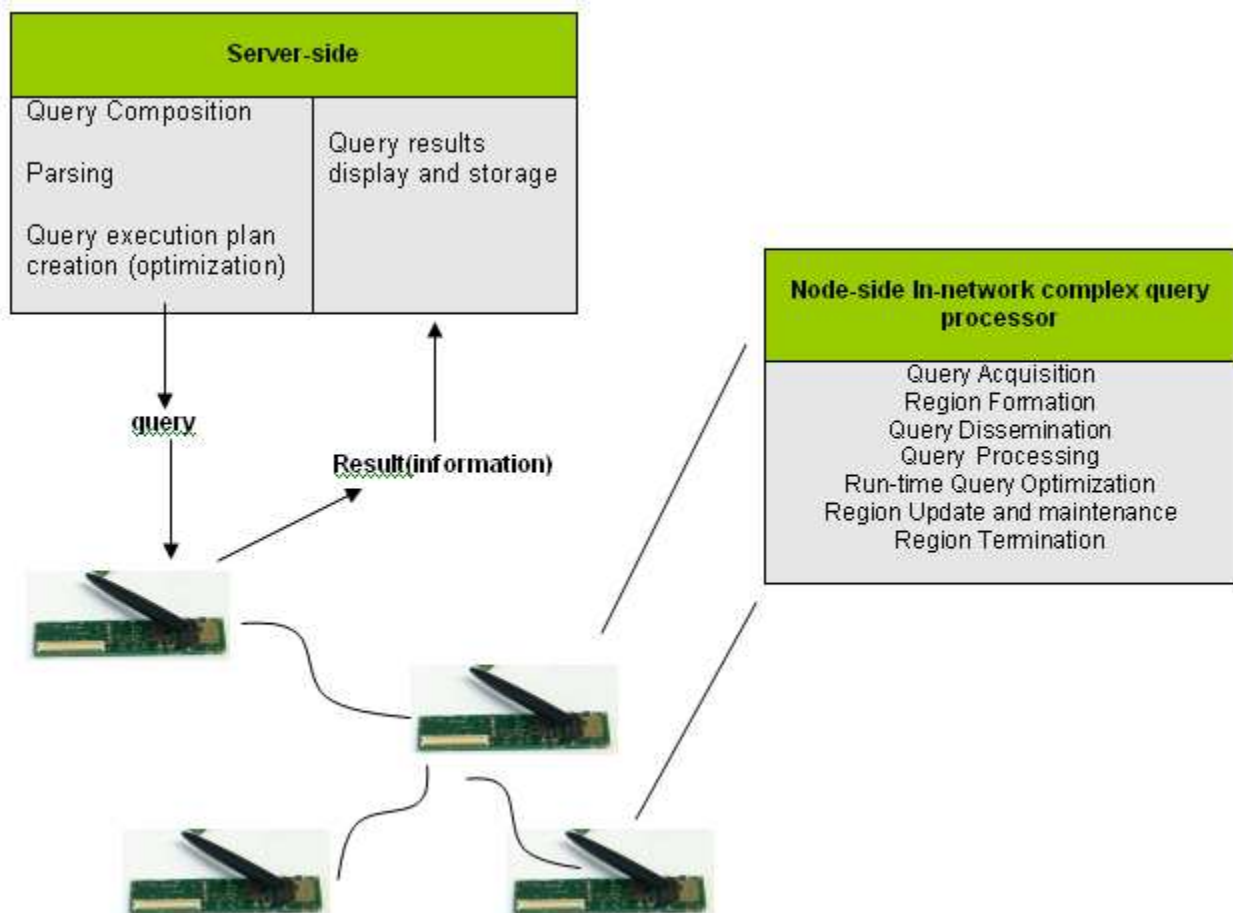
Figure 4.4: A detail view of the functions provided by the complex query processing system.

A region management service will also be running on nodes in an effort to update region information periodically as well as remove expired region data if necessary.

Two categories of complex query will be addressed in the implementation.

The first is an event-based reactive query which will form part of a monitoring system and will focus on dynamic, autonomous region formation as a result of event triggering. The idea here is that an event occurs and queries are issued as a result.

The second will be a continuous mapping query which aims to produce condensed maps for delivery to an end user. Such a query will be initiated by the user and will lead to further queries issued by the user for more fine-grain detail perhaps based on a smaller area within the network. Initial investigation of this type of query is presented in Chapter 4, Section 4.1.

Chapter 5

Experiments and Results

As a first step in the research, the feasibility of using in-network abstractions for complex query processing was examined. Two sets of experiments were carried out. The first examined the feasibility of region-based query resolution in comparison to a number of popular alternative techniques. These were all implemented and tested in simulation, and the efficiency of the in-network abstraction approach analysed against results where centralized and in-network aggregation approaches were used. The second experiment analysed the region-based approach vis-a-vie a real-life WSN deployment. In that deployment some in-network processing was carried out with all processed data being transmitted to a sink node. The experimental setup was duplicated in simulation and a number of metrics for comparing efficiency of the processing approach used in the field as opposed to the region based approach was carried out. These experiments, their results and analysis are presented in this chapter.

5.1   The Simulation Environment - SenSor

All simulations presented in this chapter were carried out using SenSor [82, 81], an in-house WSN simulator. Sensor is a scalable Python based package that allows prototyping of WSN algorithms.

The simulator allows a user to separate out the concerns of a simulation. For instance, a simulation of a communication protocol may require that packets be realistically modeled whereas for a fault management system packet design is irrelevant and can be described in less detail. Sensors are modeled as a pool of concurrent threads with each sensor being able to gather and process data from the environment and locate and communicate with its neighbours. Figure 5.1 shows a screenshot of a SenSor simulation in progress. A full description of the simulation environment and its capability has been presented in a number of publications [82, 81]. The full text of these publications is included in Appendix A.
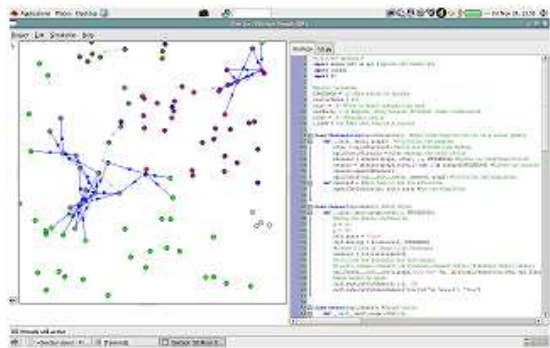
Figure 5.1: SenSor Simulator

5.2 Comparison of Query Processing Methods - Simulation Scenario and Setup

As described in Section 4.4, the particular class of WSN-based monitoring applications motivates this research work. The forest fire detection and monitoring application described is one such example as it requires in-network information extraction for extending and improving the monitoring system beyond the literature reported state-of-the-art. This motivating application formed the basis for the preliminary simulations performed with the aim of investigating the feasibility of in-network abstractions for complex query processing.

The assumption is that a wireless sensor network has been deployed to monitor a forest where events of interest have been pre-defined and are triggered if pre-conflagration gases are detected above a defined threshold. In the simulation, this threshold is set at an arbitrary value and a gas detection event is triggered at a randomly selected node once the simulation starts. Subsequently, a query is issued to attempt to generate a map of the fire if indeed the fire exists.

The simulations were allowed to run until a response to the query was returned to the sink. In general, the simulations aimed to answer two questions:

1. *Is it possible to use in-network abstractions (regions) to facilitate query processing?*

2. *How does the region-based approach compare in terms of cost against the centralized and in-network aggregation approaches currently in use?*

Query resolution is simulated using three approaches: region-based resolution, spanning tree-based in-network aggregation and centralized resolution. These are described in sections 4.3, 4.4 and 4.5 respectively. Question (1) above was investigated in three stages. First, schemes for region establishment

based on an attribute of interest were devised and implemented. Second, management of the established regions was designed and enabled in the simulation using elected region leaders. Third, query processing was coordinated within regions and not network-wide (as would be the case with the other approaches). Question (2) which in essence addresses the feasibility of the region-based approach involved conducting a quantitative comparison of the region-based approach with the commonly used spanning-tree aggregation and centralized approaches. This was done by measuring and comparing the efficiency of the three approaches in terms of the number of data messages (data packets) transmitted in the resolution of the posed query.

For all simulations, a number of controls were maintained:

1. The number of nodes for each simulation set was consistent (networks with 50, 100 and 150 were assessed for each querying approach). Nodes were scattered randomly over a two-dimensional plane measuring 600*600 units.

2. The number of runs was constant for each individual simulation (5). (Since sensors were randomly placed, the communication cost measurements differed among runs.) Multiple runs where an average could be taken were seen as a way of keeping the experiment's results as realistic as possible.

3. All nodes were initialized with an equal and consistent amount of energy.

4. Radio broadcast range was set at a 20 unit radius.

5. Each node's location was unique within the two-dimensional plane.

6. The same routing protocol was used for all simulations. Energy usage as a result of the protocol in use was not considered, rather, only the energy expended in query resolution (again this was measured in terms of the number of messages sent during query resolution was measured). Note that future work will involve an informed choice of the most appropriate routing protocol energy-wise.

7. Gas and temperature readings for all nodes were randomly generated using the same random number generator.

The results of the simulations are described and analysed in Section 5.3.

5.2.1 Region-based Querying

In order to simulate the region-based approach, the network is divided into logical regions based on a node's location within the 2-D simulation area and the regions used to facilitate query processing. In the simulation, regions are manually configured prior to query dissemination and processing, a constraint which will have to be relaxed in future work, particularly when examining dynamic region formation based on event detection. For purposes of the simulation a number of assumptions are made:

1. The cost of region setup is ignored at this point but its effect will be considered and analysed in future work.

```
.....

class aSensor(api.Sensor): #Sensor object
    def __init__(self,graph,ether,i):


        self.power=100          # a node's initial energy
        self.regionmembers=[]   #a list of region members for a node
        self.regionID='-'       #region to which a node belongs
        self.state = "Waiting"  #node's initial state
        self.isLeader = False   #indicates whether a node is a region leader or not
        self.myLeader = '-'     #Stores the id of the node who is this node's region leader


        self.pathToRegionLeader='-'    # stores a path to the node's region leader

        self.regionRoutingTable=[]  #a routing table to members of a node's region

.....
```

Figure 5.2: Code snippet of node software structure.

2. Regions are static for the duration of the simulation; however, the aim for future work is to investigate the use of dynamic regions which may change while query processing is occurring. Although not an issue in this simulation, since nodes are stationary and regions are based on geographic location, in cases where regions are established over an attribute that is changing with time like temperature, for example, the need for facilitating and managing dynamic regions becomes critical to query processing.

3. Each node is a member of a maximum of one region. Again, future research will have to investigate nodes as members of multiple regions which can be an issue in facilitating multiple queries simultaneously or processing complex queries containing multiple attributes of interest.

From an architectural standpoint, this simulation focused on performing preliminary investigations into the region formation and maintenance features of the node-side software proposed and described in Chapter 3, Section 3.3.3. A code description of the nodes included a number of data structures for storing and maintaining the information required for both region formation and region operation and maintenance. Figure 5.2 shows a code snippet of the node's structure.

The region-based querying approach demonstrated in simulation comprises two phases. These are the *Region Setup* and *Querying* phases.

Region Setup Phase

Region Discovery and Formation

An initial message is sent from the sink to the network. This message contains region setup information identifying a node as being in a particular region. A node receiving the message sets its region ID attribute and forwards the setup message on. Once all nodes within broadcast range have received the message the leader election phase begins. Visually, node colour is used to differentiate regions in the simulator. Nodes which are outside broadcast range and do not receive the setup message are not considered in the region formation process. Figure 5.3 shows a sequence diagram of the region discovery process.
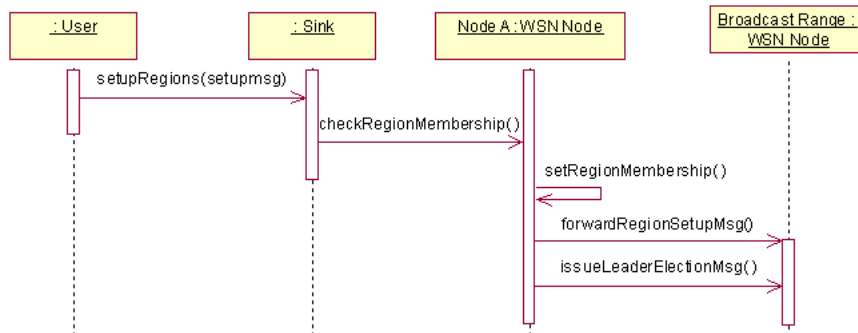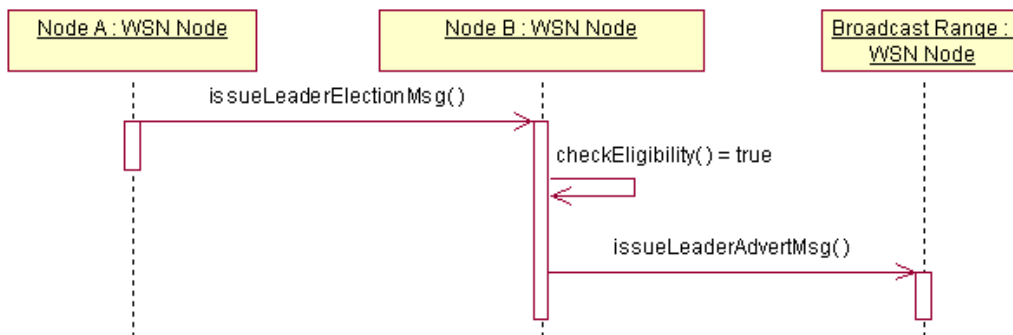
Figure 5.3: Region discovery and formation

Figure 5.4: Leader Election.

Although all nodes are identical, for simplification nodes in the diagram are named to more clearly identify its role during the simulation.

In the simulation, region setup occurs prior to a query being issued. This approach is seen as useful in applications where long term monitoring of established attributes of interest is the objective of the WSN deployment. Future work will investigate query-based region setup where regions are distinguished as part of the query itself with an indicated region life if necessary.

Leader Election

Once a node has established its region membership it issues a *leader election message*. A naïve selection method was used to elect region leaders where a node receiving an election message set itself as leader if no leader existed for the particular region. That node then sends a registration message to the sink and issues a *leader advertisement message* to the network. This message contains the leader's ID and the region ID for which it is the leader. Figure 5.4 shows a sequence diagram of the leader election process. Again all nodes are identical but for clarity of a node's role during the process, nodes are named in the diagram.

Alternative leader election schemes need to be investigated perhaps those based on relevant node characteristics such as energy level, processing power, etc.
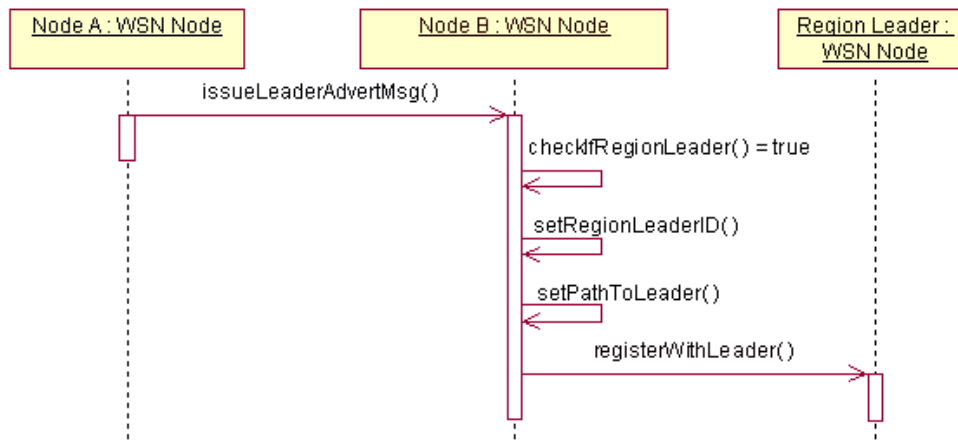
Figure 5.5: Leader Advertisement

Leader Advertisement

A node receiving a *leader advertisement message* sets its leader attribute to the leader ID in the message if they are part of the same region. Once a node has set its leader attribute it sends a registration message out to the leader along the reverse path of the advertisement message. Figure 5.5 shows a sequence diagram of the leader advertisement process.

At the end of the Region Setup phase:

1. There is at least one region established based on nodes' 2-D locations within the simulator window and hop distance between nodes.

2. All nodes within broadcast range are part of a region.

3. Each region has elected a leader.

4. Each region leader has a path to the sink, a list of its region members and a path to each member.

5. Each region member knows its leader and has a path to that leader.

Querying Phase

A gas event is simulated as a task running on nodes within the simulation. The node to first detect this 'event' immediately sends an alert message to its region leader. The leader now begins gathering the necessary data on the distribution of the gas in the vicinity of the event. The region knows the location of its members, so it sends a query to the subset of nodes within one hop of the node which detected the event. It also queries those nodes about the temperature detected in those areas.

Nodes send their gas and temperature data to the region leader for analysis. The leader decides whether the data indicates that a fire has started or not. For simplicity, in the simulation this was done using simple over the threshold, maximum temperature and gas readings. If the results indicate that a fire has been
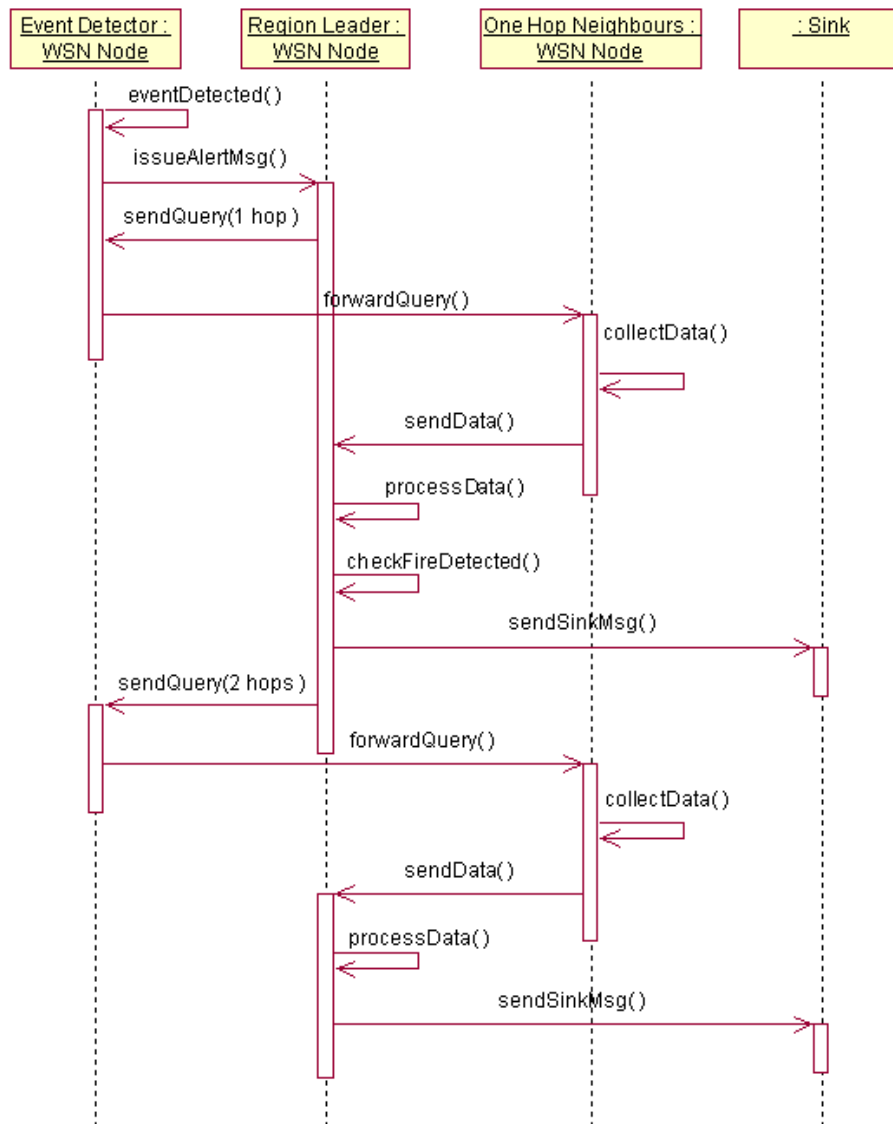
Figure 5.6: Region-based Querying

detected, the leader queries further (2 hops and then 3) to try to determine how far the fire has spread. If there is no fire detected by this initial query, the leader still queries further (2 hops) to make sure that there is indeed no fire.

The region then either sends a simple 'FALSE ALARM' message or a 'FIRE DETECTED' response containing the gas and temperature distribution over the area covered by the answering nodes (this represents a temperature/gas map of the area) to the sink. The leader continues to monitor its region for further development if a fire has been indicated and waits for input from the sink. If no message is received from the sink within a certain period of time it stops querying for gas and temperature data. In the simulation, once a response was sent to the sink, no other messages were sent to the region leader. Figure 5.6 shows a sequence diagram illustrating this phase of the simulation.

In future work, the idea of the sink issuing further queries will be investigated as well as the problem of a fire having spread to areas under survey by nodes not in the region where the event was initially detected. This would involve some form of inter-region communication or the sink needing to issue queries to the region leader of such a node.

The number of messages generated in getting the query to relevant region members, processing the query and sending that data back to the leader, and also sending a response from the leader to the sink was measured.

The results are presented in figures 5.11, 5.12 and 5.13 and discussed in section 5.4.

5.2.2 In-network Aggregation Simulation

This simulation represents the class of query-based systems that incorporate in-network processing via aggregation of data up a spanning tree of the network, to the root (sink node). The following assumptions are made:

1. The network discovery phase has already occurred and a spanning tree of the entire network has been established.

2. The cost of establishing the spanning tree is ignored.

As is the case with the region-based simulation above, a gas event is simulated and the node detecting it immediately informs the sink by sending to it an alert message. The user cannot specify any geographical information so all data generated has to be sent from node to parent until it eventually reaches the sink. All data is sent to the sink.

The sink, upon receiving the alert message, issues a query for the gas and temperature readings. When a node receives the message it forwards to its parent node a packet containing the gas and temperature readings. Data is aggregated (packet merging) wherever possible as messages move up the tree, and sent in the same data packet.

Once the sink has received this aggregate data, analysis is performed to determine if the data indicates a possible fire or not. Again, as is the case with the region-based approach a simple method of calculating
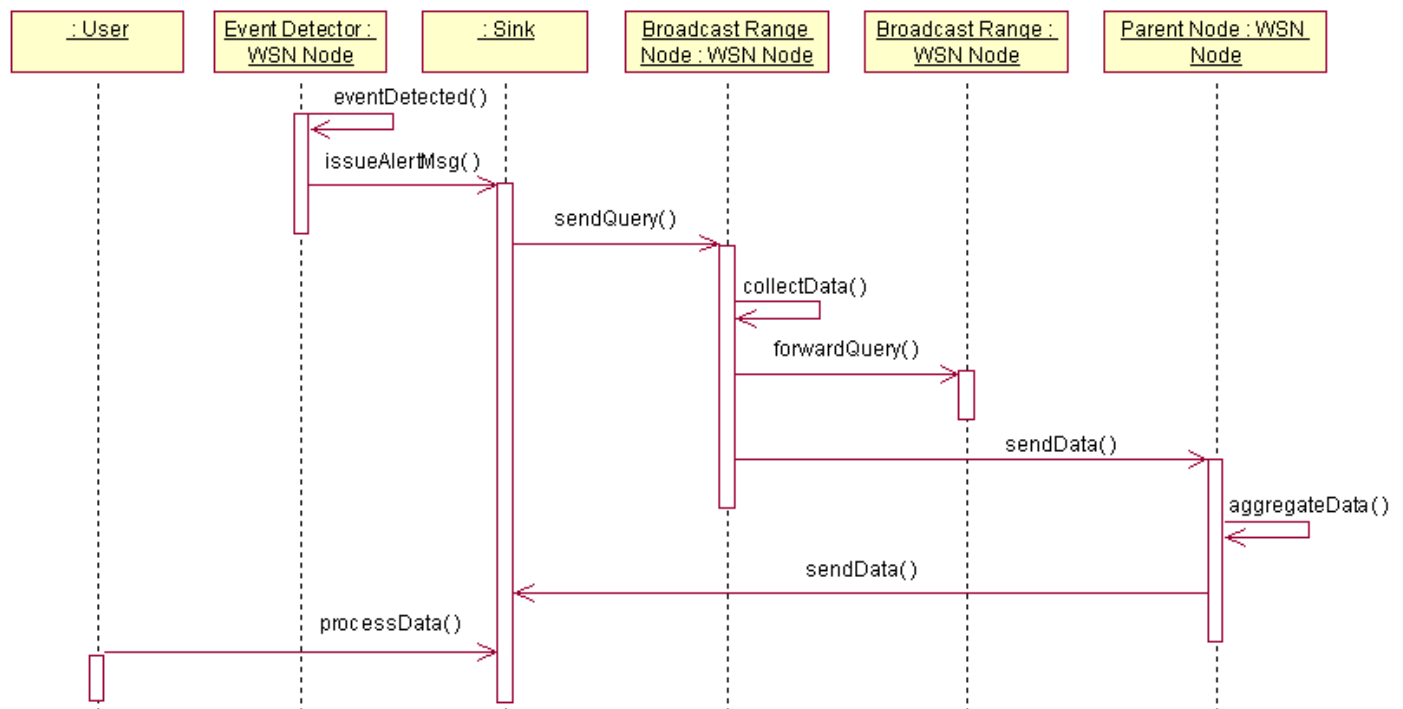
Figure 5.7: Querying with in-network aggregation

the maximum temperature and gas readings was carried out. If the results indicate a fire the sink could issue further queries, for example, a continuous query for monitoring the temperature and gas readings over the area covered by the network for a particular period of time. However, for purposes of the simulation once a response was sent to the sink the simulation was terminated. Figure 5.7 illustrates this process.

The cost of query resolution was measured by counting the number of messages generated in sending data as a result of the query back to the sink. Multiple runs on 50-node, 100-node and 150-node networks were carried out. The results are presented in figures 5.11, 5.12 and 5.13 and discussed in section 5.4.

5.2.3   Centralised Simulation

This simulation represents the basic warehousing approach where all data is sent directly to the sink for analysis. The assumption is that the network discovery phase has already occurred.

As before, a gas event is simulated and the node detecting it immediately informs the sink. Similar to the in-network aggregation approach, all data generated has to be sent back to the sink for analysis. Each node in the simulation knows a path to the sink node as a result of network discovery.

The sink upon receiving the message issues a query for the gas and temperature readings. When a node receives the message it returns to the sink a packet containing the gas and temperature readings. The
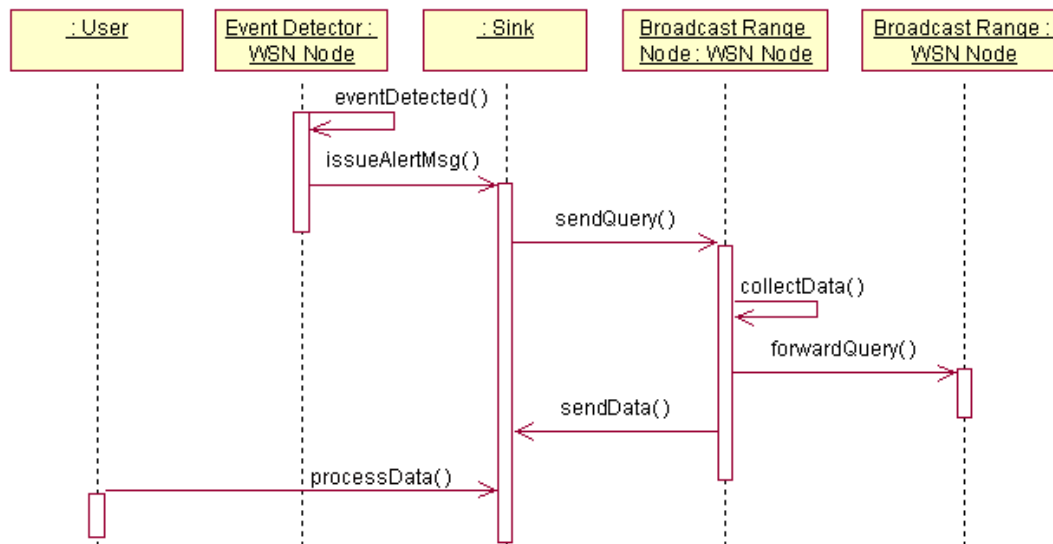
Figure 5.8: Centralized Querying

message reaches all nodes within broadcast range with nodes subsequently responding to the sink. Once the sink has received the data, analysis is performed to determine whether the data indicates that a fire has started or not. Figure 5.8 illustrates the centralized querying process.

Again, as is the case with the region-based and aggregation approaches a simple method of calculating the maximum temperature and gas readings was carried out. If the results indicate a fire the sink could issue further queries, for example, a continuous query for monitoring the temperature and gas readings over the area covered by the network for a particular period of time. However, for purposes of the simulation once a response was sent to the sink the simulation was terminated.

The cost of query resolution was measured by counting the number of data messages generated as a result of the query and sent back to the sink. Again, multiple runs on the 50-node, 100-node and 150-node networks were carried out. The results are presented in figures 5.11, 5.12 and 5.13 and discussed in section 5.4.

5.3 Acoustic Monitoring using Region-based Querying

Habitat monitoring (already described in Chapter 4, Section 4.1) is already a rich area of research in the area of WSNs particularly because of the benefits to science and education. Within this group are applications based on acoustic sensing, which are concerned with event detection and classification as well as monitoring and localization. Bioacoustics research is a specific example and acoustic sensing in that context can be used to help scientists acquire acoustic data which can can then be used to distinguish between animals, species and census counts.

The processing required for such applications usually involves complex signal processing operations and
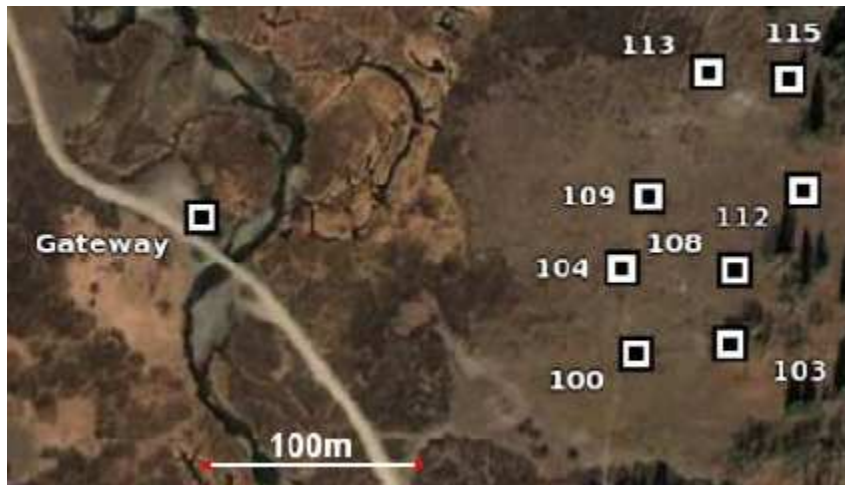
Figure 5.9: Map of VoxNet deployment area

therefore present a number of challenges and requirements that are not evident in traditional monitoring systems. The challenges include the heavy computation needed due to very high data rates and the need for development of specific algorithms to facilitate online processing of very large amounts of data.

One such acoustic monitoring application, VoxNet focused on creating a system that allowed the detection of marmot alarm calls. The work was informed by the requirements of scientists and their desire to be able to detect these marmots in the field and then localize their positions. These calls, therefore, were used to help determine the location of the animal at the time the call was detected, relative to known burrow locations. Although for some systems simple recording and offline analysis of the data fulfills application requirements, in the case of this bioacoustic monitoring system it was important that the system produce timely results. The acoustic event detection and localization application consisting of eight nodes was deployed over an area of about 9800 sq metres (2.4 acres). A gateway node was then positioned about 200m away from the nearest node. A map of the deployment is displayed in figure 5.9

5.3.1   Localisation of Acoustic Signals - the VoxNet approach

Once an acoustic signal is detected by a node it is timestamped and an attempt made to determine the location of the sound. The localization algorithm used, the Approximated Maximum Likelihood algorithm (AML) [5]functions as follows:

1. A stream of audio is processed through a 'fast path' detector to identify possible alarm calls.

2. The algorithm then estimates bearing to the caller from multiple points.

3. The algorithm then fuses those estimates to produce an estimate of the sound location.

The algorithm was expressed as a WaveScript program which is a logical dataflow graph of stream operators connected by streams. Once the program runs on these input streams results are streamed back

to data storage components, in this case the gateway or sink node. This execution of the WaveScript program constitutes the extent of in-network processing with the result being an estimate of the direction of arrival (DOA) of the acoustic signal. Timestamped, DOA values are then sent on by nodes detecting acoustic events to the sink. In some cases, depending on time availability, AML may not be executed on the node to produce DOA estimates and instead streams of raw detections are sent instead. The size of a DOA packet is 800 bytes as compared to 32KB for a raw detection.

In the application a minimum of 3 acoustic signals or DOA estimates are needed to localize a sound. At the sink, these DOA estimates along with the timestamps are used to determine, first, whether the signals are indeed from the same acoustic event and if they are, they can be combined to determine the location at which the sound occurred.

For purposes of this work, metrics from the VoxNet deployment representing the number of data packets transmitted in localizing one event, the number of hops over which these were sent as well as the total distance traveled were used. A comparison was then made between the real-life deployment results and the results of the simulations incorporating in-network, region-based processing.

5.3.2   A Region-based Approach to Acoustic Signal Localization

The author maintains that an approach that incorporated in-network processing using the concept of dynamic regions would be just as effective, if not more, than sending all raw data or partially processed data, that is, DOA estimates, back to the gateway or sink for analysis. In this set of experiments, given the limitations of the simulator, estimation of query processing times and generation of random events were not possible. Instead, metrics not dependent on time like packet size, hop count and number of packets transmitted were used to compare the efficiency and feasibility of the approach as compared to that taken in the VoxNet application in the field.

5.3.3   Simulation Scenario and Setup

For all simulations, a number of controls were maintained:

1. The deployment of nodes in the simulator mirrored the actual real-life deployment topology. Nine nodes including the sink were positioned in the simulation window at the same relative locations as they were positioned in the actual deployment (see figure 5.9.)

2. All nodes were initialized with an equal and consistent amount of energy.

3. Radio broadcast range was set at a 200 unit radius (a unit corresponding to a meter). This was the broadcast range of nodes in the VoxNet deployment.

4. Each node's location was unique within the two-dimensional plane.

5. A list of acoustic events, the ID of nodes which should detect these events and the intervals at which these events are to occur was defined prior to the start of each simulation. Lists contained between 3 and 5 events.

Region Based Query Resolution

A continuously running, event detection task (referencing the event list) is implemented on all nodes. At each time interval (a system clock tick), each node checks the event list. If an event for that node exists and the current time matches the scheduled event an acoustic event is triggered.

The detecting node logs the time of the event and checks if it has any related inquiries in its 'inquiry cache'. An inquiry indicates that another node has previously detected a possibly correlated event and has requested and is possible awaiting a response. If a node with an inquiry message in its inquiry cache detects a *related* event (that is, it is within the valid time slot) it sends a response to the node that sent the inquiry. If the node detecting the event does not have any inquiries, it sets itself as a region leader and sends an *inquiry* message to its one hop neighbours. This inquiry message contains the ID of the node that has detected the event and the event's timestamp. A node receiving this message registers the inquiry if it has not had an event that matches that inquiry. A matching or related event is one that falls within a timeslot which was determined practically through using the simulator. If the node detecting the event does have an inquiry, it checks to see if its event has occurred within the *valid time slot*[1] and a 'DATA' message is transmitted to the node who sent the inquiry message (the region leader). Any event occurring out of the valid time slot will not be sent to the region leader in response to its inquiry message.

The region leader upon receipt of a DATA message, records that message and if it has already received the minimum (3) required or more sends the result to the sink via a RESULT message. In the simulation, if the region leader has already received 3 messages it still waits for a period of time before sending a result on. The reason is that the greater the number of messages the better the accuracy of the measurement (this is also the model used in the VoxNet system.) The time a region leader waits again was determined through use of the simulator although in the real life deployment this value was determined experimentally and is referred to as a fuzz factor.

The simulation was allowed to run until either a result was sent to the sink or if the time for detection of a particular event had elapsed. For example, an insufficient number of detections (less than three) were detected within the valid timeslot. The results of this simulation in comparison to the in-the-field results are analysed in section 5.4.

5.4   Results and Analysis

5.4.1   Efficiency of Region-based Querying

The results obtained indicate that regions **can** be used to support in-network query processing. In analyzing how much more efficient, if at all, the region-based approach is as compared to the in-network aggregation and centralized approaches, energy efficiency is evaluated using one parameter, that is, the number of messages generated in returning a query response to the sink.

The results of running a number of 50-node simulations indicate that on average region-based query pro-

---

[1]A time range is used to indicate whether two or more signals are correlated and therefore considered as relating to the same acoustic event. This is referred to as the valid time slot.
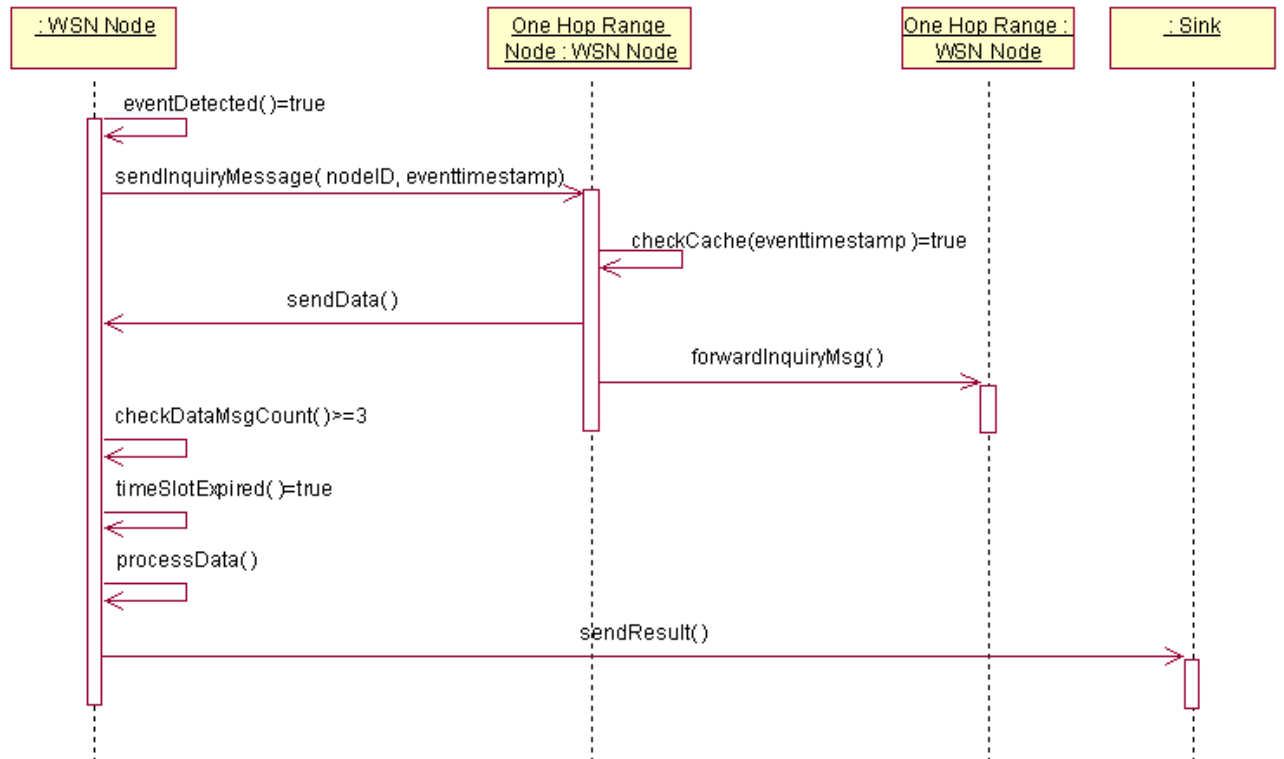
Figure 5.10: Region-based approach to processing of acoustic data.
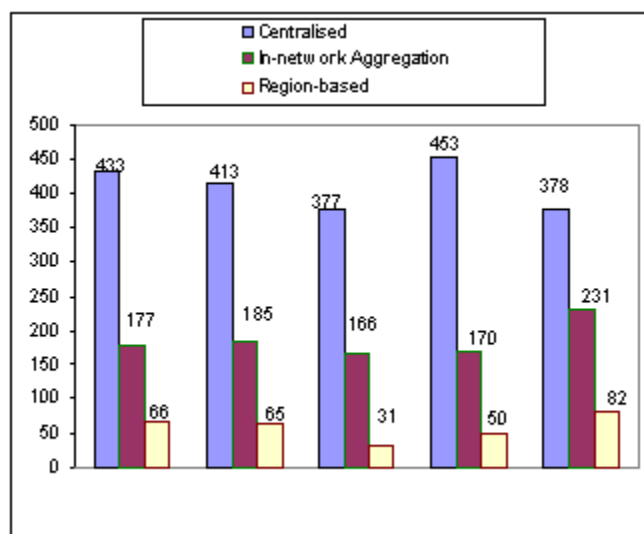

Figure 5.11: Comparison of number of messages required for query resolution in a 50-node network.
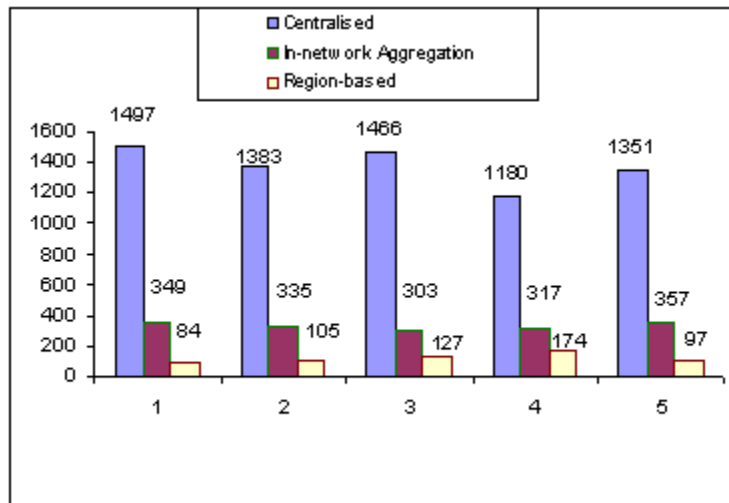
Figure 5.12: Comparison of number of messages required for query resolution in a 100-node network.

cessing resulted in an almost 86% decrease in the number of transmissions over the centralized approach and a 68% decrease on average over the approach using in-network aggregation.

For a 100-node simulation the results were equally impressive with over 91% reduction over the centralized approach and an almost 65% decrease when compared to the simulation using in-network aggregation. For a 150-node simulation the results for the region-based approach were more marked when compared to the centralized approach with over 95% reduction in the number of messages while there was an over 45% decrease when compared to the simulation using in-network aggregation.

The results although showed that the efficiency of the region-based querying approach increased as the size of the network increased when compared to the centralized approach, going from 86% to 91% to 95% for a 50, 100 and 150 node network respectively. In comparison to the in-network aggregation approach, however, this was not the case. Although communication was less, the relative percentages showed an overall decrease, going from 68% to 65% to 45% for a 50, 100 and 150 node network respectively. It would have been interesting to run these algorithms on even larger networks in determining whether this trend would continue, however, the limitation of the simulator used made this impossible at this time. Based on these results, however, although feasible, the scalability of the region-based approach is an issue to be considered and examined more closely in future work.

5.4.2   Effectiveness of Region-based Querying

Again, the results obtained indicate that regions **can** be used to support in-network query processing and are a viable approach in the context of a real-life deployment scenario. In analyzing the effectiveness and feasibility of the region-based processing approach compared to that used in the VoxNet application a number of measures were taken. First, the number of data packets was measured and along with the packet size was used to calculate the total data transmission required for query resolution. This was
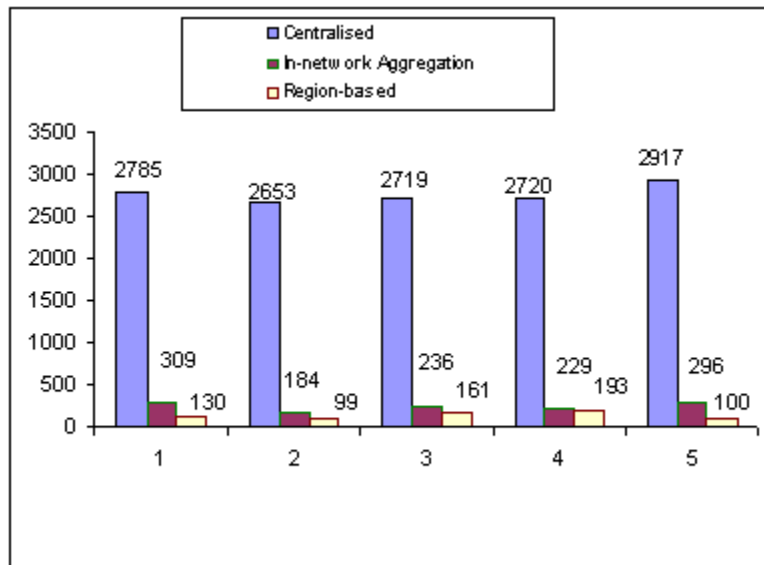
Figure 5.13: Comparison of number of messages required for query resolution in a 150-node network.

done in multiple runs for cluster/region sizes of 3 and 4 nodes in the case of the VoxNet and region-based approaches respectively, and an average taken. In addition, the average total data transported was calculated for the VoxNet system in scenarios where all raw data was sent to the sink and also in cases where in-network processing for DOA estimation was carried out.

The results, displayed in figure 5.14, clearly indicate that in terms of data transmission savings the region-based approach exhibits an advantage over the query processing approach used in the VoxNet system. This was evident even when some in-network processing was carried out. For a 3 and 4 node region there was approximately a 28% and 38%, reduction respectively in the amount of data transmitted over the VoxNet approach with in-network processing. The decrease is even more marked with figures of 61% and 67% for the VoxNet processing approach with no in-network analysis.

The simulation again confirmed the feasibility of the approach although a number of issues need to be investigated in future work. One, as identified before, is the scalability of the approach. In this simulation the network was quite small (9 nodes) and the regions created as a consequence also limited in size. An interesting exercise for the future would be to investigate the approach using larger regions and also to test the region query processing algorithm in the field.

Figure 5.14: Total amount of data transmitted in query resolution

Chapter 6

Conclusions and Future Work

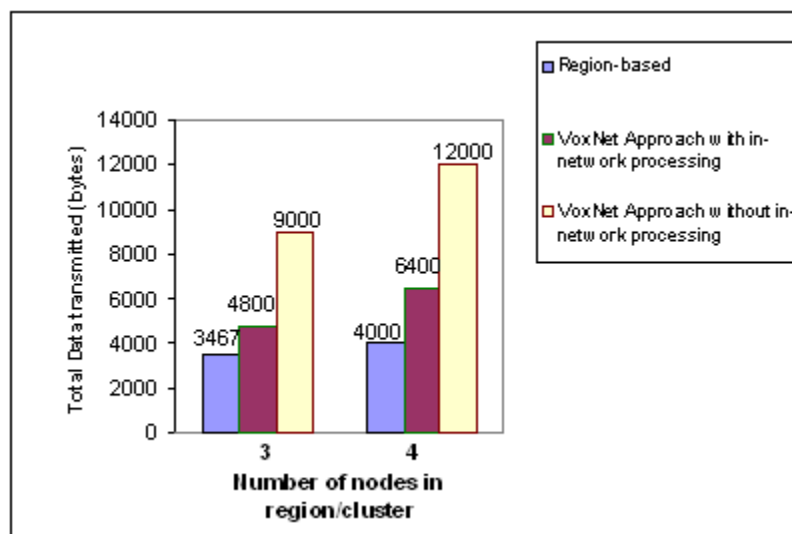An extensive literature review has shown the absence of complex query mechanisms in the query processing systems in use today. The review did highlight, however, applications in which these queries could be used along with in-network processing to both extend and improve WSNs' deployment value.

The research work so far has demonstrated the need for the complex queries described in the report and the feasibility of using the proposed logical abstractions (called regions) to facilitate query processing in WSN systems. The results of preliminary experimentation showed that ultimately the region-based approach was more energy efficient than the currently popular approaches to in-network query processing. Further, an architecture for a distributed complex query processing system has been developed and preliminary experiments have already begun on investigating processing techniques and developing particular aspects of the proposed system.

The preliminary experiments have produced promising results but also highlighted a number of areas that have to be considered and investigated in future research work before progress can be made. A number of these are described below.

The work put forward has already indicated that the region construct is part of the query itself and drives both query dissemination and processing. The simulations looked at query processing where regions had already been implemented. Future work will have to investigate the issues involved in query-based region setup.

In the simulation, all regions created were static for the duration of the simulation; the use of dynamic regions which can change while query processing is occurring needs to be investigated as well. This becomes extremely important in cases where regions are established over an attribute that is changing with time like temperature.

Although not considered at this point, the cost of region setup and its effect will need to be considered and analysed in future work. Combining it with cluster formation to reduce energy consumption is perhaps one approach.

For simplicity, the simulation only considered a scenario where a node could only be a member of a maximum of one region. The research, however, will have to investigate nodes as members of multiple

regions. This can be an issue in facilitating the processing of multiple queries simultaneously or processing complex queries containing subqueries with additional attributes of interest.

In the simulation, once a response was sent to the sink the querying process terminated. In future work, the idea of the sink issuing further queries will be investigated as well as inter-region communication and how it can be facilitated.

In terms of a query language for region-based processing, a candidate SQL-based language has already been identified. Future work will seek to amend and extend the language to allow the construction of the queries of interest identified in the report.

Finally, candidate queries have already been identified as 'test cases' for a prototype implementation of the DCQP. A number of measures have already been identified for testing and evaluating the success of the system.

Bibliography

[1] AID-N. Aid-n project, 2007.

[2] K. Akkaya and M. Younis. A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, 3(3):325–49, 2005. Copyright 2005, IEE 8535173 routing protocol wireless sensor network network flow quality of service energy-aware routing protocol classification QoS.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002. Copyright 2002, IEE 7257257 wireless sensor networks microelectro-mechanical systems wireless communications digital electronics communication architecture ad hoc networks application layer transport layer networking layer routing data link layer medium access control error control physical layer power aware protocols.

[4] J. N. Al-Karaki and A. E. Kamal. Routing techniques in wireless sensor networks: a survey. *IEEE Wireless Communications*, 11(6):6–28, 2004. Copyright 2005, IEE 8239261 1536-1284 routing technique wireless sensor network routing protocol network structure flit routing hierarchical routing location-based routing multipath-based protocol query-based protocol negotiation-based protocol QoS-based protocol quality of service coherent-based protocol.

[5] A. M. Ali, Yao Kung, T. C. Collier, C. E. Taylor, D. T. Blumstein, and L. Girod. An empirical study of collaborative acoustic source localization. Proceedings of the Sixth International Symposium on Information Processing in Sensor Networks, pages 41–50, Cambridge, MA, USA, 2007. IEEE. 9660598 approximate maximum likelihood estimation high frequency signals spatial aliasing pre-recorded source signal DOA accuracy direction-of- arrival accuracy marmot alarm-calls AML-based source localization algorithm collaborative acoustic source localization.

[6] Argo. Argo: part of the integrated global observation strategy., 2007.

[7] Anish Arora, Rajiv Ramnath, Emre Ertin, Prasun Sinha, Sandip Bapat, Vinayak Naik, Vinod Kulathumani, Hongwei Zhang, Hui Cao, Mukundan Sridharan, Santosh Kumar, Nick Seddon, Chris Anderson, Ted Herman, Nishank Trivedi, Chen Zhang, Mikhail Nesterenko, Romil Shah, Sandeep Kulkarni, Mahesh Aramugam, Limin Wang, Mohamed Gouda, Young-Ri Choi, David Culler, Prabal Dutta, Cory Sharp, Oilman Tolle, Mike Grimmer, Bill Ferriera, and Ken Parker. Exscal: Elements of an extreme scale wireless sensor network. Proceedings - 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pages 102–108, Hong Kong, China, 2005. Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ

08855-1331, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 064010148654 Wireless sensor network Software platforms Peer to peer ad hoc networks.

[8] S. Baydere, M.A. Ergin, and O. Durmaz. constructing wireless sensor networks via effective topology maintenance and querying. In *Third Annual Mediterranean Ad Hoc Networking Workshop, Med-Hoc-Net 2004*, Bodrum, Turkey, 2004.

[9] Guo Bin, Li Zhe, and Meng Yan. A dynamic-clustering reactive routing algorithm for wireless sensor networks. First International Conference on Communications and Networking in China, ChinaCom '06, page 4149783, Beijing, China, 2007. Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ 08855-1331, United States. Compilation and indexing terms, Copyright 2008 Elsevier Inc. 074610922093 Dynamic clustering reactive routing algorithms Power distribution.

[10] R. Bird. *Introduction to Functional Programming using Haskell*. Prentice Hall, New York, 1998.

[11] P. Bonnet, J. Gehrke, and P. Seshadri. Querying the physical world. *IEEE Personal Communications*, 7(5):10–15, 2000. Copyright 2000, IEE 6756441 1070-9916 physical world querying sensors small-scale mobile devices processors memory monitoring applications centralized system device data collection device database system distributed query execution techniques communication reduction distributed query processing meta-data management device networks COUGAR Device Database Project.

[12] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. Mobile Data Management. Second International Conference, MDM 2001. Proceedings (Lecture Notes in Computer Science Vol.1987), pages 3–14, Hong Kong, China, 2001. Springer-Verlag. 7017396 sensor database systems sensor networks surveillance applications long-running queries stored data sensor data centralized system raw data time series long-running query persistent view COUGAR system.

[13] Philippe Bonnet and Praveen Seshadri. Device database systems. *Proceedings - International Conference on Data Engineering*, page 194, 2000. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 00255165455 Device database systems.

[14] Athanassios Boulis, Chih-Chieh Han, Roy Shea, and Mani B. Srivastava. Sensorware: Programming sensor networks beyond code update and querying. *Pervasive and Mobile Computing*, 3(4):386–412, 2007. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 072510658831 Wireless ad hoc sensor networks Distributed programming Sensor nodes.

[15] Athanassios Boulis, Chih-Chieh Han, and Mani B. Srivastava. Design and implementation of a framework for efficient and programmable sensor networks, 2003. 1066121 187-200.

[16] David Braginsky and Deborah Estrin. Rumor routing algorithm for sensor networks. Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications, pages 22–31, Atlanta, GA, United States, 2002. Association for Computing Machinery. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 03087365422 Routing algorithms.

[17] Phil Buonadonna, David Gay, Joseph M. Hellerstein, Wei Hong, and Samuel Madden. Task: Sensor network in a box. volume 2005 of *Proceedings of the Second European Workshop on Wireless*

Sensor Networks, EWSN 2005, pages 133–144, Istanbul, Turkey, 2005. Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ 08855-1331, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 063210055195 Sensor networks Sensornet systems Tiny Application Sensor Sensornet research.

[18] Iacopo Carreras, Imrich Chlamtac, Hagen Woesner, and Honggang Zhang. Nomadic sensor networks. volume 2005 of *Proceedings of the Second European Workshop on Wireless Sensor Networks, EWSN 2005*, pages 166–175, Istanbul, Turkey, 2005. Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ 08855-1331, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 063210055198 Wireless Sensor Networks (WSN) Sensor node design Nomadic sensor networks Parking scenario.

[19] Mauricio Castillo-Effen, Daniel H. Quintela, Ramiro Jordan, Wayne Westhoff, and Wilfrido Moreno. Wireless sensor networks for flash-flood alerting. Proceedings of the IEEE International Caracas Conference on Devices, Circuits and Systems, ICCDCS, pages 142–146, Dominican Republic, 2004. Institute of Electrical and Electronics Engineers Inc., Piscataway, NJ 08855-1331, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 05509536262 Flash-flood alerting Wireless sensor networks Design constraints System-level design.

[20] D. Chamberlin. Xquery: An xml query language. *IBM Systems Journal*, 41(4):597–615, 2002. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 03047329968 0018-8670 Expressions.

[21] Ioannis Chatzigiannakis, Georgios Mylonas, and Sotiris Nikoletseas. Jwebdust: A java-based generic application environment for wireless sensor networks. volume 3560 of *Lecture Notes in Computer Science*, pages 376–386, Marina del Rey, CA, United States, 2005. Springer Verlag, Heidelberg, D-69121, Germany. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 05429417839 Wireless sensor networks Control center Integrated management.

[22] Canfeng Chen, Jian Ma, and Ke Yu. Designing energy-efficient wireless sensor networks with mobile sinks, Oct. 31- Nov. 3, 2006 2006.

[23] M. Chen, T. Kwon, Y. Yuan, and V.C.M. Leung. Mobile agent based wireless sensor networks. *Journal of Computers*, 1(1):14–21, 2006.

[24] Shen Chien-Chung, C. Srisathapornphat, and C. Jaikaeo. Sensor information networking architecture and applications. *IEEE Personal Communications*, 8(4):52–9, 2001. Copyright 2001, IEE 7029831 1070-9916 sensor information networking architecture sensor information networking applications sensor network monitoring sensor network tasking middleware sensor network nodes massively distributed objects SINA execution environment configuration primitives communication primitives scalable organization energy-efficient organization programmable substrate declarative queries programming script wireless computing sensor programming language sensor query and tasking language mobile computing interworking.

[25] Atish Datta Chowdhury and Shivashankar Balu. Consensus: A system study of monitoring applications for wireless sensor networks. Proceedings - Conference on Local Computer Networks,

LCN, pages 587–588, Tampa, FL, United States, 2004. IEEE Computer Society, Los Alamitos, CA 90720-1314, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 05269183740 Functionality Wireless sensor networks (WSN) Monitoring applications Application-defined clusters.

[26] Maurice Chu, Horst Haussecker, and Feng Zhao. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *International Journal of High Performance Computing Applications*, 16(3):293–313, 2002. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 063810122344 1094-3420 Constrained anisotropic diffusion routing (CADR) Information-driven sensor querying (IDSQ) Node failures Sensor networks.

[27] CodeBlue. Codeblue: Wireless sensor netoworks for medical care, 2007.

[28] Carlo Curino, Matteo Giani, Marco Giorgetta, Alessandro Giusti, Amy L. Murphy, and Gian Pietro Picco. Mobile data collection in sensor networks: The tinylime middleware. *Pervasive and Mobile Computing*, 1(4):446–469, 2005. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 06139781204 1574-1192 Wireless sensor networks Mobile ad hoc networks Tuple space.

[29] T.E. Daniel, S. N. I. Mount, R. M. Newman, and E. I. Gaura. Towards a trusted compiler for a query language for wireless sensor networks. In *2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISOLA 2006)*, pages 277–282, Paphos, Cyprus, 2006.

[30] T.E. Daniel, S. N. I. Mount, R. M. Newman, and E. I. Gaura. Towards trusted compilation for querying in wireless sensor networks. In *Eighteenth International Conference on Systems Engineering (ICSE '06)*, pages 89–95, Coventry, United Kingdom, 2006.

[31] A. Demers, J. Gehrke, Rajaraman Rajmohan, N. Trigoni, and Yao Yong. The cougar project: a work-in-progress report. *SIGMOD Record*, 32(4):53–9, 2003. Copyright 2004, IEE 8039405 0163-5808 Cougar sensor database database approach sensor network high-level declarative language SQL variant data dissemination query processing energy-efficient routing in-network aggregation.

[32] Rasit Eskicioglu, Sayed Ahmed, and Sajid Hussain. A query processing architecture for sensor networks. In *WICON Workshop on Information Fusion and Dissemination in Wireless Sensor Networks (SensorFusion)*, Budapest, Hungary, 2005.

[33] C. L. Fok, G. C. Roman, and C. Lu. Mobile agent middleware for sensor networks: an application case study. 2005 Fourth International Symposium on Information Processing in Sensor Networks (IEEE Cat. No.05EX1086), pages 382–7, Los Angeles, CA, USA, 2005. IEEE. 8613365 mobile agent middleware wireless sensor network WSN case study Agilla local tuple space application-specific task general-purpose computing platform autonomous application fire tracking application MICA2 mote network reliability.

[34] Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. volume 2005 of *Proceedings - International Conference on Distributed Computing Systems*, pages 653–662, Columbus, OH, United

States, 2005. Institute of Electrical and Electronics Engineers Inc., Piscataway, NJ 08855-1331, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 06249933655 Adaptive wireless sensor network applications Wireless sensor networks (WSN) Pre-installed application Rich infrastructure.

[35] Thaddeus R. F. Fulford-Jones, Gu-Yeon Wei, and Matt Welsh. A portable, low-power, wireless two-lead ekg system. volume 26 III of *Annual International Conference of the IEEE Engineering in Medicine and Biology - Proceedings*, pages 2141–2144, San Francisco, CA, United States, 2004. Institute of Electrical and Electronics Engineers Inc., Piscataway, NJ 08855-1331, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 05028780278 Mica2 Mote Wireless sensor network Heart rate.

[36] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks. SenSys'03: Proceedings of the First International Conference on Embedded Networked Sensor Systems, pages 138–149, Los Angeles, CA, United States, 2003. Association for Computing Machinery, New York, NY 10036-5701, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 05229125946 Sensor networks Time synchronization Packet delay Clock drift Medium access control (MAC).

[37] Johannes Gehrke and Samuel Madden. Query processing in sensor networks. *IEEE Pervasive Computing*, 3(1):46–55, January - March 2004.

[38] Tolle Gilman, Polastre Joseph, Szewczyk Robert, Culler David, Turner Neil, Tu Kevin, Burgess Stephen, Dawson Todd, Buonadonna Phil, Gay David, and Hong Wei. A macroscope in the redwoods, 2005. 1098925 51-63.

[39] Omprakash Gnawali, Ramesh Govindan, and John Heidemann. Implementing a sensor database system using a generic data dissemination mechanism. *IEEE Data Engineering Bulletin*, 28(1):70–75, 2005.

[40] R. Govindan, J. M. Hellerstein, W. Hong, S. Madden, M. Franklin, and S. Shenker. The sensor network as a database. Technical 0-771, Computer Science Dept., University of Southern California., 2000.

[41] R. Gummadi, O. Gnawali, and R. Govindan. Macro-programming wireless sensor networks using kairos. Distributed Computing in Sensor Systems. First IEEE International Conference, DCOSS 2005. Proceedings (Lecture Notes in Computer Science Vol. 3560), pages 126–40, Marina del Rey, CA, USA, 2005. Springer-Verlag. 8522228 wireless sensor network macro-programming Kairos sensor network programming global behavior distributed computation compile-time subsystem run-time subsystem programming primitives distributed-code generation distributed-code instantiation remote data access remote data management inter-node program flow coordination programming model distributed programs infrastructure services signal processing tasks routing tree construction localization object tracking.

[42] Gumstix. Gumstix, 2007.

[43] P. Gupta and P. R. Kumar. The capacity of wireless networks. *Information Theory, IEEE Transactions on*, 46(2):388–404, 2000.

[44] Richard Guy, Ben Greenstein, John Hicks, Rahul Kapur, Nithya Ramanathan, Tom Schoellhammer, Thanos Stathopoulos, Karen Weeks, Kevin Chang, Lew Girod, and Deborah Estrin. Experiences with the extensible sensing system ess. Technical 61, CENS, 2006. 1052213.

[45] Carl Hartung, Richard Han, Carl Seielstad, and Saxon Holbrook. Firewxnet: A multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments. volume 2006 of *MobiSys 2006 - Fourth International Conference on Mobile Systems, Applications and Services*, pages 28–41, Uppsala, Sweden, 2006. Association for Computing Machinery, New York, NY 10036-5701, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 063910139380 Wireless Sensor Networks Forest Fires Deployments Weather conditions.

[46] John Heidemann, Fabio Silva, and Deborah Estrin. Matching data dissemination algorithms to application requirements. SenSys'03: Proceedings of the First International Conference on Embedded Networked Sensor Systems, pages 218–229, Los Angeles, CA, United States, 2003. Association for Computing Machinery, New York, NY 10036-5701, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 05229125953 Sensor networks Data dissemination Network routing Directed diffusion Application performance.

[47] John Heidemann, Fabio Silva, Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, and Deepak Ganesan. Building efficient wireless sensor networks with low-level naming. volume 35 of *Operating Systems Review (ACM)*, pages 146–159, Banff, Alta., Canada, 2002. Association for Computing Machinery. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 02397101398 Wireless sensor networks Low level naming In network processing Data aggregation Network topology Attribute based naming system.

[48] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. volume vol.2 of *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, page 10 pp. vol.2, Maui, HI, USA, 2000. IEEE Comput. Soc. Also available on CD-ROM in PDF format 6530466 energy-efficient communication protocol wireless distributed microsensor system reliable monitoring civil applications military applications energy dissipation LEACH Low-Energy Adaptive Clustering Hierarchy clustering-based protocol randomized local cluster based station rotation energy load distribution localized coordination scalability robustness dynamic networks data fusion routing protocol simulations useful system lifetime.

[49] J. M. Hellerstein, Hong Wei, S. Madden, and K. Stanek. Beyond average: toward sophisticated sensing with queries. Information Processing in Sensor Networks. Second International Workshop, IPSN 2003. Proceedings (Lecture Notes in Computer Science Vol.2634), pages 63–79, Palo Alto, CA, USA, 2003. Springer-Verlag. 7977390 high-level query languages sensor networks embedded programming distributed programming TinyDB sensornet query engine topographic mapping wavelet-based compression vehicle tracking.

[50] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS, pages 93–104, Cambridge, MA, 2000. Association for Computing Machinery. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 01276567461 Networked sensors.

[51] Masayuki Hirafuji, Tokihiro Fukatsu, Takuji Haoming, Tominari Watanabe, and Seishi Ninomiya. A wireless sensor network with field-monitoring servers and metbroker in paddy fields. Proceedings - World Rice Research Conference, Tokyo and Tsukuba, Japan, 2004.

[52] HPWREN. High performance wireless research and education network, 2007.

[53] http://jade.tilab.com. Jade, 2006.

[54] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. MobiCom 2000. Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking, pages 56–67, Boston, MA, USA, 2000. ACM. 6957287 directed diffusion scalable robust communication sensor networks distributed sensing environmental phenomena data communication application-aware nodes caching paths remote surveillance.

[55] Chalermek Intanagonwiwat, Deborah Estrin, Ramesh Govindan, and John Heidemann. Impact of network density on data aggregation in wireless sensor networks. Proceedings - International Conference on Distributed Computing Systems, pages 457–458, Vienna, Austria, 2002. Institute of Electrical and Electronics Engineers Inc. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 02387092643 Greedy aggregation.

[56] Ali Iranli, Morteza Maleki, and Massoud Pedram. Energy efficient strategies for deployment of a two-level wireless sensor network. Proceedings of the International Symposium on Low Power Electronics and Design, pages 233–238, San Diego, CA, United States, 2005. Institute of Electrical and Electronics Engineers Inc., Piscataway, NJ 08855-1331, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 05509536481 Sensor deployment Wireless sensor network Collinear deployment Planar deployment.

[57] Nauman Israr and Irfan Awan. Multihop clustering algorithm for load balancing in wireless sensor networks. International Journal of Simulation: Systems, Science and Technology, 8(3):13–25, 2007.

[58] D. N. Jayasimha, S. Sitharama Iyengar, and R. L. Kashyap. Information integration and synchronization in distributed sensor networks. IEEE Transactions on Systems, Man and Cybernetics, 21(5):1032–1043, 1991. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 92050469903 0018-9472 Distributed Sensor Networks Sensor Information Integration Information Synchronization.

[59] Rajgopal Kannan, Sudipta Sarangi, and S. Sitharama Iyengar. Sensor-centric energy-constrained reliable query routing for wireless sensor networks. Journal of Parallel and Distributed Computing, 64(7):839–852, 2004. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 05018763184 0743-7315 Wireless sensor networks Reliable query routing Sensor centric routing Routing algorithms.

[60] H. Karl and A. Willig. A short survey of wireless sensor networks. Technical TKN-03-018, Telecommunication Network Group, Technische Universitat, 2003.

[61] BjÃ¶rn Karlsson, Oscar BÃ€ckstrÃ¶m, Wlodek Kulesza, and Leif Axelsson. Intelligent sensor networks: An agent-oriented approach. In *REALWSN*, Stockholm, Sweden, 2005.

[62] E. Katsiri and A. Mycroft. Knowledge representation and scalable abstract reasoning for sentient computing using first-order logic. In *1st Workshop on Challenges and Novel Applications for Automated Reasoning, in conjunction with CADE-19*, pages p. 73–82, Miami FL, 2002.

[63] B. Krishnamachari, D. Estrin, and S. Wicker. Modelling data-centric routing in wireless sensor networks. In *IEEE InfoCom*, 2002.

[64] Koen Langendoen, Aline Baggio, and Otto Visser. Murphy loves potatoes experiences from a pilot sensor network deployment in precision agriculture. *20th International Parallel and Distributed Processing Symposium, IPDPS 2006*, 2006:1639412, 2006. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 070910440499 Large-scale sensor network Development process Wireless sensor network (WSN).

[65] Philip Levis and David Culler. Mate: A tiny virtual machine for sensor networks. International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS, pages 85–95, San Jose, CA, United States, 2002. Association for Computing Machinery. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 03057345730 Virtual machine Sensor networks Software Package TinyOS.

[66] Jerome Peter Lynch, Kincho H. Law, Anne S. Kiremidjian, Thomas Kenny, and Ed Carryer. A wireless modular monitoring system for civil structures. volume 4753 I of *Proceedings of SPIE - The International Society for Optical Engineering*, pages 1–6, Los Angeles, CA, United States, 2002. The International Society for Optical Engineering. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 02477228835 Wireless modular monitoring systems.

[67] S. Madden and M. J. Franklin. Fjording the stream: an architecture for queries over streaming sensor data. Proceedings 18th International Conference on Data Engineering, pages 555–66, San Jose, CA, USA, 2002. IEEE Comput. Soc. 7232155 query architecture streaming sensor data wireless networks environmental monitoring data collection intermittent connectivity power constraints periodic sampling Fjords architecture multiple query management sensor resource demands query throughput traffic sensor network traces Interstate 80 inter-state highway traffic performance communication costs power consumption.

[68] S. Madden, M. J. Franklin, J. M. Hellerstein, and Hong Wei. Tag: a tiny aggregation service for ad hoc sensor networks. Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02), pages 131–46, Boston, MA, USA, 2002. USENIX Assoc. 7797360 Tiny AGgregation ad hoc network sensor network TAG service low-power environment distributed computing wireless environment declarative query low-power sensor wireless sensor performance evaluation fault tolerance.

[69] S. Madden, W. Hong, and Carlos Guestrin. Tinydb, November 2007 2007.

[70] S. Madden, R. Szewczyk, M. J. Franklin, and D. Culler. Supporting aggregate queries over ad-hoc wireless sensor networks. Proceedings Fourth IEEE Workshop on Mobile Computing Systems and

Applications, pages 49–58, Callicoon, NY, USA, 2002. IEEE Comput. Soc. 7387973 aggregate queries ad-hoc wireless sensor networks relational database TinyOS operating system generic query interface data aggregation data reduction tool SQL optimization.

[71] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1):122–73, 2005. Copyright 2005, IEE 8532158 0362-5915 TinyDB acquisitional query processing system sensor networks SQL distributed query processor power consumption.

[72] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. The design of an acquisitional query processor for sensor networks. Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 491–502, San Diego, CA, United States, 2003. Association for Computing Machinery, New York, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 04098040770.

[73] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless sensor networks for habitat monitoring. Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications, pages 88–97, Atlanta, GA, United States, 2002. Association for Computing Machinery. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 03087365429 Wireless sensor networks.

[74] D. Marsh, R. Tynan, D. O'Kane, and G. M. P. O'Hare. Autonomic wireless sensor networks. *Engineering Applications of Artificial Intelligence*, 17(7):741–8, 2004. Copyright 2005, IEE 8356361 0952-1976 autonomic wireless sensor networks autonomic computing multiagent systems environmental nervous system distributed agents agent-oriented software.

[75] Kirk Martinez, Royan Ong, and Jane Hart. Glacsweb: A sensor network for hostile environments. 2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, IEEE SECON 2004, pages 81–87, Santa Clara, CA, United States, 2004. Institute of Electrical and Electronics Engineers Inc., New York, NY 10016-5997, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 05259167227 Sensor networks Radio communications Low power Environmental monitoring Glaciology.

[76] Daniel Massaguert, Chien-Liang Fok, Nalini Venkatasubramanian, Gruia-Catalin Roman, and Chenyang Lu. Exploring sensor networks using mobile agents. volume 2006 of *Proceedings of the International Conference on Autonomous Agents*, pages 323–325, Hakodate, Japan, 2006. Association for Computing Machinery, New York, NY 10036-5701, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 071710568645 Multi-agent planning Performance evaluation Agent systems Programming abstractions.

[77] Mate. Mate: Building application-specific sensor network language runtimes, 2003.

[78] Gokhan Mergen, Qing Zhao, and Lang Tong. Sensor networks with mobile access: Energy and capacity considerations. *IEEE Transactions on Communications*, 54(11):2033–2044, 2006. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 064910282566 Sensor network with mobile access (SENMA) Multihop ad hoc architecture Mobile access points (APs) Multiaccess channels.

[79] Michael Mertsock and David Stawski. Wireless sensor nodes as intelligent agents: Using expert systems with directed diffusion, 2005.

[80] Chen Min, Kwon Taekyoung, and Choi Yanghee. Data dissemination based on mobile agent in wireless sensor networks. volume 2005 of *Proceedings - Conference on Local Computer Networks, LCN*, pages 527–528, Sydney, Australia, 2005. IEEE Computer Society, Los Alamitos, CA 90720-1314, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 064810277354 Data dissemination Wireless sensor networks Energy awareness Mobile agents.

[81] S. N. I. Mount, R. M. Newman, and E. I. Gaura. A simulation tool for system services in ad-hoc wireless sensor networks. In *NSTI Nanotechnology Conference and Trade Show (Nanotech '05)*, volume 3, Ch. 7, pages 423–426, Anaheim, California, USA, 2005.

[82] S. N. I. Mount, R. M. Newman, E. I. Gaura, and J. Kemp. Sensor: an algorithmic simulator for wireless sensor networks., 2006 2006.

[83] S. N. I. Mount, R. M. Newman, S. R. Lakin, R. J. Low, and E. I. Gaura. Asque: An agent communication language for ad-hoc wireless sensor networks. volume 2005 of *Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference*, pages 404–411, Porto, Portugal, 2005. Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ 08855-1331, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 063410081744 Network communication Ad-hoc wireless sensor networks Agent communication languages.

[84] A. L. Murphy, G. P. Picco, and G. C. Roman. Lime: a coordination model and middleware supporting mobility of hosts and agents. *ACM Transactions on Software Engineering and Methodology*, 15(3):279–328, 2006. Copyright 2007, The Institution of Engineering and Technology 9495483 1049-331X Linda middleware tuple space formal semantic characterization coordination model mobility management mobile agent mobile computing.

[85] S. Nath, Y. Ke, P. B. Gibbons, B. Karp, and S. Seshan. Irisnet: An architecture for enabling sensor-enriched internet service. Technical IRP-TR-03-04, Intel Research, June 2003 2003.

[86] Grape Networks. Grape networks' climate genie, 2007.

[87] Ryan Newton. *Compiling Functional Reactive Macroprograms for Sensor Networks*. Masters, Massachusetts Institute of Technology, 2005.

[88] Ryan Newton, Arvind, and Matt Welsh. Building up to macroprogramming: An intermediate language for sensor networks. volume 2005 of *2005 4th International Symposium on Information Processing in Sensor Networks, IPSN 2005*, pages 37–44, Los Angeles, CA, United States, 2005. Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ 08855-1331, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 06249933679 Design space Token Machine Language (TML) Lower-level runtime environment Atomic action model of concurrency.

[89] Ryan Newton, Greg Morrisett, and Matt Welsh. The regiment macroprogramming system, 2007. 1236422 489-498.

[90] Ryan Newton and Matt Welsh. Region streams: functional macroprogramming for sensor networks, 2004. 1052213 78-87.

[91] G.M.P. O'Hare, D. Marsh, A. Ruzzelli, and R. Tynan. Agents for wireless sensor network power management. In *International Workshop on Wireless and Sensor Networks (WSNET-05)*, Oslo, Norway, 2005. IEEE Press.

[92] K. Pister. The 29 palms experiment - tracking vehicles with a uav-delivered sensor network, 2007.

[93] Riccardo Poli and William B. Langdon. Backward-chaining evolutionary algorithms. *Artificial Intelligence*, 170(11):953–982, 2006. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 062910008046 0004-3702 Genetic programming Efficient search Backward chaining Tournament selection.

[94] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–8, 2000. Copyright 2000, IEE 6599253 0001-0782 wireless integrated network sensors computing performance densely distributed networks Internet access embedded sensors embedded controls embedded processors monitoring transportation industry manufacturing industry health care environmental oversight safety security microsensor technology low-power signal processing computation low-cost wireless networking compact system.

[95] L. Prasad, S. S. Iyengar, R. L. Kashyap, and R. N. Madan. Functional characterization of sensor integration in distributed sensor networks. Proceedings. The Fifth International Parallel Processing Symposium (Cat. No.91TH0363-2), pages 186–93, Anaheim, CA, USA, 1991. IEEE Comput. Soc. Press. 4091468 distributed sensor networks fault-tolerant integration abstract interval estimates sensor output output interval estimate.

[96] H. Qi, S. Iyengar, and K. Chakrabarty. Multiresolution data integration using mobile agents in distributed sensor networks. *Systems, Man and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 31(3):383–391, 2001. 1094-6977.

[97] H. Qi, S. S. Iyengar, and K. Chakrabarty. Distributed multi-resolution data integration using mobile agents. volume vol.3 of *2001 IEEE Aerospace Conference Proceedings (Cat. No.01TH8542)*, pages 3–1133, Big Sky, MT, USA, 2001. IEEE. 7074394 mobile agents distributed multiresolution data integration improved infrastructure distributed sensor network processing code data locations network bandwidth network latency distributed integration problem optimum performance problem data transfer time sensor fusion.

[98] H. Qi, Xiaoling Wang, and K. Chakrabarty. Multisensor data fusion in distributed sensor networks using mobile agents. In *5th International Converence on Information Fusion*, Annapolis, MD., 2001.

[99] Hairong Qi and Wang Feiyi. Optimal itinerary analysis for mobile agents in ad hoc wireless sensor networks. volume vol.1 of *Wireless 2001. 13th International Conference on Wireless Communications. Proceedings*, pages 147–53, Calgary, Alta., Canada, 2001. TRLabs/Univ. Calgary. 7637369

itinerary optimization mobile agents ad hoc networks wireless sensor networks data fusion client-server paradigm distributed sensor networks network bandwidth computation engine routing.

[100] Hairong Qi, Xiaoling Wang, S. Sitharama Iyengar, and Krishnendu Chakrabarty. High performance sensor integration in distributed sensor networks using mobile agents. *International Journal of High Performance Computing Applications*, 16(3):325–335, 2002. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 063810122345 1094-3420 Distributed sensor networks Mobile agents High performance distributed computing Sensor integration.

[101] Hairong Qi, Yingyue Xu, and Xiaoling Wang. Mobile-agent-based collaborative signal and information processing in sensor networks. *Proceedings of the IEEE*, 91(8):1172–1183, 2003. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 05289200452 0018-9219 Collaborative signal and information processing (CSIP) Mobile-agent-based computing Execution time Sensor networks.

[102] K. Ramamohanarao, L. Kulik, S. Selvadurai, B. Scholz, U. Roehm, E. Tanin, A. Viglas, A. Zomaya, and C. Leckie. A survey on data processing issues in wireless sensor networks for enterprise information infrastructure, May 2006.

[103] S. Ramanathan and M. Steenstrup. A survey of routing techniques for mobile communications networks. *Journal of Special Topics in Mobile Networks and Applications (MONET)*, 1(2):89–104, 1996. Copyright 1996, IEE 5445927 1383-469X routing techniques mobile communications networks mobile wireless networks routing system design roving users network topology fluctuating link quality node mobility.

[104] ROADNet. Real-time observatories, applications and data management network, 2007.

[105] Kay Romer, Oliver Kasten, and Friedemann Mattern. Middleware challenges for wireless sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):59–61, 2002. 643556.

[106] Kenneth N. Ross, Ronald D. Chaney, George V. Cybenko, Daniel J. Burroughs, and Alan S. Willsky. Mobile agents in adaptive hierarchical bayesian networks for global awareness. volume 3 of *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 2207–2212, San Diego, CA, USA, 1998. IEEE, Piscataway, NJ, USA. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 99024553606 Mobile agents Bayesian networks.

[107] L. B. Ruiz, J. M. Nogueira, and A. A. F. Loureiro. Manna: a management architecture for wireless sensor networks. *IEEE Communications Magazine*, 41(2):116–25, 2003. Copyright 2003, IEE 7538077 0163-6804 MANNA management architecture wireless sensor networks environmental monitoring infrastructure management public safety office security transportation military application medical application home application computing devices Internet network nodes MANNA management architecture WSN agents physical management architectures WSN management management information base.

[108] Narayanan Sadagopan, Bhaskar Krishnamachari, and Ahmed Helmy. Active query forwarding in sensor networks. *Ad Hoc Networks*, 3(1):91–113, 2005. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 04538756417 1570-8705 Flooding based querying (FBQ) Expanding ring search (ERS) Vehicle tracking Energy constrained sensors.

[109] SCADDS. Tinydiffusion, 2007.

[110] R. Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer archi-tectures and applications. Proceedings First International Conference on Peer-to-Peer Computing, pages 101–2, Linkoping, Sweden, 2002. IEEE Comput. Soc. 7226733 peer-to-peer networking peer-to-peer architectures client server-architectures networking concepts Internet.

[111] Praveen Seshadri. Predator: A resource for database research. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 27(1):16–20, 1998. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 04057875126 0163-5808.

[112] Rahul C. Shah, Sumit Roy, Sushant Jain, and Waylon Brunette. Data mules: Modeling and analysis of a three-tier architecture for sparse sensor networks. *Ad Hoc Networks*, 1(2-3):215–233, 2003. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 05159030234 Sensor networks Mobility Random walk Three-tier architecture.

[113] Li Shuoqi, S. H. Son, and J. A. Stankovic. Event detection services using data service middleware in distributed sensor networks. Information Processing in Sensor Networks. Second International Workshop, IPSN 2003. Proceedings (Lecture Notes in Computer Science Vol.2634), pages 502–17, Palo Alto, CA, USA, 2003. Springer-Verlag. 7977417 event detection services data service middleware distributed sensor networks DSWare data semantics failure rate group-based services data-centric false alarms.

[114] Adam Silberstein. Push and pull in sensor network query processing. In *Southeast Workshop on Data and Information Management (SWDIM '06)*, Raleigh, North Carolina, 2006.

[115] K. Sohrabi, J. Gao, V. Ailawadhi, and G. J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, 7(5):16–27, 2000. Copyright 2000, IEE 6756442 1070-9916 protocols algorithms static nodes constrained energy resources slow mobility energy-efficient routing ad hoc subnetworks cooperative signal processing self-organizing wireless sensor networks routing protocols MAC protocols medium access control mobile ad hoc networks cellular networks short range wireless local area networks wireless LAN.

[116] Byungrak Son, Yong-sork Her, and Jung-Gyu Kim. A design and implementaiton of forest-fires surveillance system based on wireless sensor networks for south korea mountains. *International Journal of Computer Science and Network Security*, 6(9B):124–130, 2006.

[117] Ryo Sugihara and Rajesh K. Gupta. Programming models for sensor networks: A survey. *ACM Transactions on Sensor Networks*, 4(2), 2008.

[118] Kim Sukun, Pakzad Shamim, Culler David, Demmel James, Fenves Gregory, Glaser Steven, and Turon Martin. Health monitoring of civil infrastructures using wireless sensor networks, 2007. 1236395 254-263.

[119] Leo Szumel, Jason LeBrun, and John D. Owens. Towards a mobile agent framework for sensor networks. volume 2005 of *Second IEEE Workshop on Embedded Networked Sensors, EmNetS-II*, pages 79–87, Sydney, Australia, 2005. Institute of Electrical and Electronics Engineers Computer

Society, Piscataway, NJ 08855-1331, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 06259953669 Reprogramming Mobile agent framework Sensor networks.

[120] Liu Ting and M. Martonosi. Impala: a middleware system for managing autonomic parallel sensor systems. *SIGPLAN Notices*, 38(10):107–18, 2003. Copyright 2004, IEE 7976963 0362-1340 Impala autonomic parallel sensor wireless transceivers middleware architecture wireless sensor networks application adaptation software system ZebraNet HP/Compaq iPAQ Pocket PC handhelds Pocket PC version simulations mobile sensor system deployment autonomous parallel systems software adaptation software update.

[121] R. Tynan, G. M. P. O'Hare, D. Marsh, and D. O'Kane. Multi-agent system architectures for wireless sensor networks. Computational Science - ICCS 2005. 5th International Conference. Proceedings, Part III (Lecture Notes in Computer Science Vol. 3516), pages 687–94, Atlanta, GA, USA, 2005. Springer-Verlag. 8614335 multiagent system architectures wireless sensor networks networked devices constrained resources.

[122] Richard Tynan, David Marsh, Donal O'Kane, and G. M. P. O'Hare. Intelligent agents for wireless sensor networks. Proceedings of the International Conference on Autonomous Agents, pages 1283–1284, Utrecht, Netherlands, 2005. Association for Computing Machinery, New York, NY 10036-5701, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 06119758635 Sensor nodes Power management scheme Sensor platforms.

[123] Richard Tynan, Antonio Ruzzelli, and G.M.P. O'Hare. A methodology for the deployment of multi-agent systems on wireless sensor networks. In *17th International Conference on Software Engineering and Knowledge Engineering, (SEKE '07)*, Taiwan, China, 2005. IJSEKE press.

[124] V. K. Vaishnavi, Q. Wu, N. S. V. Rao, H. Qi, K. Chakrabarty, S. S. Iyenger, and J. Barhen. On computing mobile agent routes for data fusion in distributed sensor networks. *IEEE Transactions on Knowledge and Data Engineering*, 16(6):740–53, 2004. Copyright 2004, IEE 7999245 1041-4347 mobile agent routing problem data fusion distributed sensor networks NP-complete problem genetic algorithm.

[125] M. Welsh and G. Mainland. Programming sensor networks using abstract regions. First Symposium on Networked Systems Design and Implementation (NSDI '04), pages 29–42, San Francisco, CA, USA, 2004. USENIX Assoc. 8434020 wireless sensor networks environmental monitoring vehicle tracking radio communication telecommunication network reliability radio connectivity geographic location TinyOS adaptive sensor network.

[126] Robert Wesson, Frederick Hayes-Roth, John W. Burge, Cathleen Stasz, and Carl A. Sunshine. Network structures for distributed situation assessment. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(1):5–23, 1981. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 81090005923 0018-9472.

[127] Kamin Whitehouse, Cory Sharp, Eric Brewer, and David Culler. Hood: A neighborhood abstraction for sensor networks. MobiSys 2004 - Second International Conference on Mobile Systems, Applications and Services, pages 99–110, Boston, MA, United States, 2004. Association for Computing

Machinery, New York, NY 10036-5701, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 04408382703 Abstraction Data sharing Distributed algorithms Neighborhood Sensor networks.

[128] Kamin Whitehouse, Feng Zhao, and Jie Liu. Semantic streams: A framework for composable semantic interpretation of sensor data. volume 3868 NCS of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 5–20, Zurich, Switzerland, 2006. Springer Verlag, Heidelberg, D-69121, Germany. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 06279983917 Semantic Streams Magnetometer data Declarative queries.

[129] Q. Wu, N. S. V. Rao, J. Barhen, S. S. Iyenger, V. K. Vaishnavi, H. Qi, and K. Chakrabarty. On computing mobile agent routes for data fusion in distributed sensor networks. *Knowledge and Data Engineering, IEEE Transactions on*, 16(6):740–753, 2004. 1041-4347.

[130] Yang Xiaoyan, Lim Hock Beng, M. Tamer, zsu, and Tan Kian Lee. In-network execution of monitoring queries in sensor networks, 2007. 1247538 521-532.

[131] Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin. A wireless sensor network for structural monitoring. SenSys'04 - Proceedings of the Second International Conference on Embedded Networked Sensor Systems, pages 13–24, Baltimore, MD, United States, 2004. Association for Computing Machinery, New York, NY 10036-5701, United States. Compilation and indexing terms, Copyright 2007 Elsevier Inc. All rights reserved 05469473071 Sensor networks Structural health monitoring Wisden.

[132] Y. Yao and J. Gehrke. Query processing for sensor networks. In *Conference on Innovative Data Systems Research (CIDR)*, 2003.

[133] Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, 2002. 601861.

[134] M. Younis, M. Youssef, and K. Arisha. Energy-aware routing in cluster-based sensor networks, 2002. 882620 129.

[135] O. Younis and S. Fahmy. Heed: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. *IEEE Transactions on Mobile Computing*, 3(4):366–79, 2004. Copyright 2004, IEE 8211585 HEED approach ad hoc sensor networks network scalability topology control approach hybrid energy-efficient distributed clustering iteration methods intercluster transmission intracluster transmission network lifetime fault tolerance minimum selection probability.

[136] Y. Yu, D. Estrin, and R. Govindan. Geographical and energy-aware routing: a recursive data dissemination protocol for wireless sensor networks. Technical UCLA-CSD TR-01-0023, UCLA Computer Science Department, May 2001 2001.

[137] Yao Yuxia, Tang Xueyan, and Lim Ee-Peng. In-network processing of nearest neighbor queries for wireless sensor networks. Database Systems for Advanced Applications. 11th International Conference, DASFAA 2006. Proceedings (Lecture Notes in Computer Science Vol.3882), pages 35–49,

Singapore, 2006. Springer-Verlag. 8923807 in-network processing nearest neighbor queries wireless sensor networks civilian applications military applications sensor nodes object tracking spatial queries energy efficiency query processing centralized data storage distributed scheme DNN energy consumption network lifetime.

Appendix A

Publications

Cogent Computing Applied Research Centre
Coventry University
Priory Street, Coventry CV1 5FB
www.cogentcomputing.org

Coventry
University